Availability by Design

A Complementary Approach to Denial-of-Service

Vigo, Roberto

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

Availability by Design

A Complementary Approach to Denial-of-Service

Roberto Vigo

Kongens Lyngby 2014
PHD-2014-353

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, Building 324,
DK-2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk PHD-2014-353

I have laid no claim to genius,
none to infallibility; but I have
endeavoured to be accurate,
and aspired to be useful.

Goold Brown, *The Grammar of English Grammars*

Summary

In computer security, a Denial-of-Service (DoS) attack aims at making a resource unavailable. DoS attacks to systems of public concern occur increasingly and have become infamous on the Internet, where they have targeted major corporations and institutions, thus reaching the general public. There exist various practical techniques to face DoS attacks and mitigate their effects, yet we witness the successfulness of many.

The need for a renewed investigation of availability gains in relevance when considering that our life is more and more dominated by Cyber-Physical Systems (CPSs), large-scale network of sensors that interact with the physical environment. CPSs are increasingly exploited in the realisation of critical infrastructure, from the power grid to healthcare, traffic control, and defence applications. Such systems are particularly prone to DoS attacks: in addition to classic communication-based attacks, their components can be subject to physical capture. Moreover, sensors are often powered by batteries, and time-limited unavailability is usually a stage planned to prolong their life span.

This dissertation argues that techniques rooted in the theory and practice of programming languages, *language-based techniques*, offer a unifying framework to deal with the *consequences* of DoS, thereby encompassing inadvertent and malicious sources of unavailability in a uniform manner.

In support to this claim we develop a family of process calculi, the Quality Calculi, where availability considerations are promoted to be first-class objects of the language domain. Moreover, these modelling tools are complemented by static analyses that pinpoint where and why unavailability may occur, levering the enhanced expressiveness of the language.

The ultimate aim of the framework is to foster the development of systems resilient to DoS by means of a principled design process, in which formal models allow, and verification tools enforce, the production of such robust code.

Resumè

Denial-of-Service (DoS) er betegnelsen for et angreb, der sigter efter at gøre en ressource utilgængelig. DoS angreb på samfundssystemer sker stadig oftere, og de er berygtede på Internettet, hvor de for nylig blev rettet mod store virksomheder. Der er forskellige metoder til at imødegå DoS angreb og nedsætte deres effekt, men mange angreb er alligevel successfulde.

Fornyet fokus på tilgængelighed er også påkrævet for systemer, som kaldes *Cyber-Physical Systems* (CPS'er). CPS'er er store netværk af sensorer og aktuatorer som interagerer med det fysiske miljø. CPSs'er bliver brugt til at bygge vigtig infrastruktur for ledningsnettet, sundhedsvæsenet, færdslen, forsvaret osv. Disse systemer er særligt følsomme overfor til DoS. Foruden de klassiske kommunicationsbaserede angreb, er deres komponenter underlagt fysisk angreb. Ydermere er komponenterne ofte drevet af batterier, og derfor kan de blive utilgængelig for at spare energi og forlænge deres levetid.

Formålet med denne afhandling er, at bevise at *sprog-baserede teknikker*, der har deres rod i det teoretiske og praktiske fundament for programmeringssprog, udgør et samlet udgangspunkt for at klare *konsekvenserne* af DoS både for utilsigtede årsager og angreb.

For at understøtte denne påstand er der udviklet en familie af proceskalkuler, de *Quality Calculi*, hvor tilgængelighed er et førsteklasses element i domænesproget. Desuden er disse modelleringsprog suppleret med statiske analyser, der kan fastslå hvilke komponeneter kan blive ikke tilgængelig og hvorfor, takket være kalkulernes udtryksfuldhed.

Hovedformålet med vores begrebsramme er at støtte udviklingen af systemer som er immune overfor DoS ved hjælp af en styret design process, hvor formelle modeller tillader, og verifikationsværktøjer håndhæver, fremstillingen af sådan robust kode.

Preface

This thesis was prepared at DTU Compute, the Department of Applied Mathematics and Computer Science of the Technical University of Denmark, in partial fulfilment of the requirements for acquiring the Ph.D. degree in Computer Science.

The Ph.D. study has been carried out under the supervision of Professor Flemming Nielson and Professor Hanne Riis Nielson in the period from December 2011 to November 2014. The Ph.D. project is funded by IDEA4CPS, a project granted by the Danish Research Foundations for Basic Research (nr. DNRF86-10).

Most of the work behind this dissertation has been carried out independently and I take full responsibility for its contents. A substantial part of the scientific work reported in this thesis is based on joint work with my supervisors [RNV12, VNR13, VNR14a, VNR14b, VNR14c], which is in part currently under submission. Within the framework of IDEA4CPS, my supervisors also fostered a collaboration with Alessio Di Mauro, Davide Papini, and Nicola Dragoni [DPVD12] (at that time all affiliated with the Technical University of Denmark), and they inspired and stimulated me to further investigate those ideas, leading to [Vig12]. Finally, we collaborated with Alessandro Celestini, Francesco Tiezzi, Rocco De Nicola [VCT+14] (at that time all affiliated with the Institute for Advances Studies - IMT Lucca, Italy). The relevance of those publications to this thesis shall be clarified in each chapter.

Beyond what mere bibliography tells, the work presented in this dissertation greatly benefited from a prolonged and fruitful interaction with a number of excellent researchers affiliated with the Technical University of Denmark, Aalborg University, and the IT University of Copenhagen, all involved in the MT-LAB research centre. Similarly, during my stay at IMT Lucca as guest scholar I had the opportunity to discuss with and learn from a great many exquisite researchers affiliated with IMT Lucca, the University of Florence, as well as a number of established and well-known computer scientists that were visiting

IMT at that time. The same applies to a number of researchers that visited the Technical University of Denmark during my study. Among them, Nikolaj Bjørner had a determining role in directing my investigation towards SAT and SMT techniques, thanks to a Ph.D. course organised by my supervisors and featuring him as invited professor.

Kongens Lyngby, November 2014
Roberto Vigo

Acknowledgements

First and foremost, I should like to thank my supervisors Flemming and Hanne. As much as I strove, I could not make up for the energy, the passion, and the dedication they put in their guidance. Let this line bear witness to my most sincere appreciation.

I would have never come to Denmark if it had not been for Gilberto Filè, who first introduced me to formal methods and since then never ceased challenging me to undertake a Ph.D. My gratitude is second only to the number of hours he devoted to my first scientific upbringing.

Thanks must go to Rocco De Nicola, who hosted me in the SysMA group at IMT Lucca, where I found a bracing environment which fostered my interaction with a handful of accomplished scientists and young brilliant researchers. In particular, I should like to mention here Alessandro Celestini, Michele Loreti, and Francesco Tiezzi, who have all proven gifted sparring partners, and whose versatility I appreciated in professional as well as in personal discussions.

I am grateful to the members of my thesis assessment committee, Alberto Lluch Lafuente, Björn Victor, and Luca Viganò for accepting to read and review this manuscript, for their valuable comments, and for they fostered a challenging yet constructive discussion during the examination.

I should like to thank current and former members of the Language-based Technology section at DTU. Ender, Kebin, and Nataliya for being my mentors in pectore. Alessandro, Marieta, and Zaruhi for trading in my productivity for memorable moments. For better or for worse, this work is dedicated to them, without whom this dissertation would have taken half of the time and would be twice as thick. Zaruhi for starting my writing days with a motivational session before spending hers painstakingly correcting my drafts. Alessandro, Lars, Laust, and Sebastian for endless, pointless, and nonetheless irresistible discussions spanning all branches of human knowledge. Lars and Laust for uncovering some of the enigmas of the Danish mentality, creating some new,

and never giving up on the idea that I could mumble some Danish. Omar for he has always had a sincere smile. Sebastian for proving talk after talk that passion does not necessarily water down with time. Christian for he has always been silently available. Cathrin for sorting out all bureaucratic bothers – the rabbit survived the trial. Alejandro, Carroline, Erisa, Fuyuan, Lijun, Michal, Piotr, and Ximeng for contributing each in their manner to creating a friendly and stimulating environment. José for he essentially had a part in all of the above: hard work and leisure, otia et negotia.

Finally, my most heart-felt thoughts are for my parents, whose example is a light unto my path, and for my siblings, who have always had the tact, the wisdom, and the fortitude to convey their sympathy in tough times through witty persiflage. I hope this tome will eventually explain to them what I have been up to for the last three years – I give up.

18.XI.2014
Roberto

Contents

List of Tables

Chapter 1

Introduction

> Besides, it has all the charm of inventing the science of navigation while already on board ship.
>
> Robin Milner on the foundations of informatics [Mil06]

1.1 Challenge

Availability is "the property of being accessible and usable upon demand by an authorised entity" [ISOb], and its absence is termed Denial-of-Service (DoS) or unavailability. Typical instances of DoS occur when the resources of a server are exhausted, preventing a given service to be offered to clients and often leading to the paralysis of an entire system, with a domino effect.

Availability is numbered among security properties, together with confidentiality and integrity, forming the so-called CIA properties. Nonetheless, it is manifest how the corpus on availability cannot compare to the literature on other security properties, neither in terms of size, nor practical effectiveness, nor theoretical insight. Despite this lesser attention, DoS attacks to systems of public concern occur increasingly and have become infamous on the Internet, the distributed system *par excellence*, where they cause enormous damages (cf. Figs. 1.1,1.2). Besides active attackers, limited resources or optimistic assumptions about the environment can be source of unavailability, suggesting that *cryptography is not the ultimate solution* to all security problems.

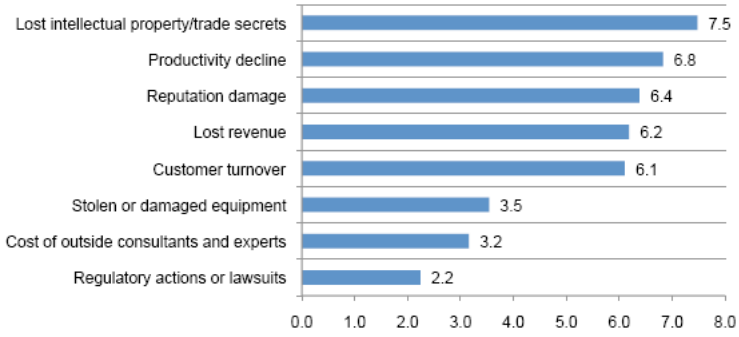


Figure 1.1: Ranking DoS damages [Pon12].

Existing literature on availability zeroes in on mechanisms to detect and avoid DoS attacks on the target side. This is a challenging task, as the detection process itself can be frustrated by the ongoing attack: to coin a provocative slogan we could say that *a detected DoS attack is a successful attack*. Moreover, the majority of techniques developed so far for confronting DoS aims at making attacks more costly (or, dually, at increasing the resource to the defender), and are oriented to facing cyber attacks such as SYN flooding. Whilst these are surely the most common availability attacks, other sources of DoS escape the protection range of this cost-based methods, and no systematic back-up solution seems to have been advanced.

Among the systems that demand for a novel, formal investigation of availability, Cyber-Physical Systems [VNR14c] (CPSs) stand out as they are more and more dominating our every-day life. These systems consist of a network of sensors and actuators that interact with the physical environment and exchange information on a cyber layer. CPSs are increasingly exploited in the realisation of critical infrastructure, from the power grid to healthcare, traffic control, and defence applications [STCE14]. Such systems are particularly prone to DoS: being composed by a great number of sensors, it is not always possible to protect their components physically. Furthermore, such components have computational and power limitations that dissuade from resorting to the cost-based approach mentioned above: it is not always feasible to increase the resources of a node. Finally, unavailability can be a legal operating status of a component, due to its duty cycle or simply to its life cycle.

1.2 Contribution

In order to tackle the challenge of developing intrinsically *robust* systems, we propose to pass from a muscular approach that blindly confronts DoS hazards, to

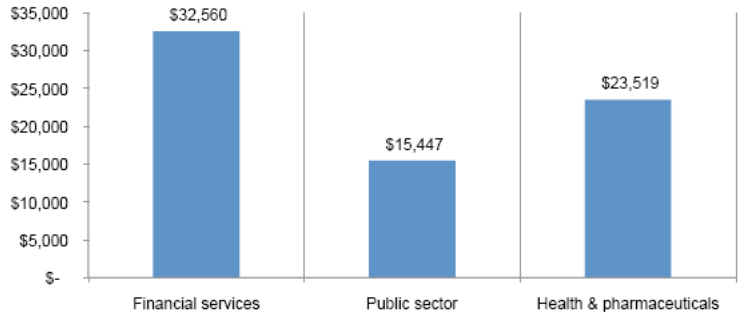


Figure 1.2: Down-time cost per minute [Pon12].

smarter *unavailability-aware* components, which follow alternative paths when their partners do not cooperate. The focus is thus lifted from a self-centred approach, wondering *how do I survive to an availability attack?*, to a perspective that considers the overall system, wondering *what do I do if my partners are unavailable?*, *perhaps* because *they* are undergoing a DoS attack. Hence, the spirit of our proposal is to cope with the *effect* of DoS, instead of confronting attacks directly, and thus aims at stopping the domino effect mentioned above.

In order to ensure availability, we advocate for a defensive programming style that avoids making benign assumptions about the surrounding environment: this pragmatic view suggests to admit the existence of DoS and try to circumvent its effects. This approach proves useful to handle availability concerns in a wide range of application domains, including software, physical, and cyber-physical systems – whatever the source of unavailability may be.

In order to support such a shift in paradigm, in this dissertation we study process calculi that promote availability concerns to be first-class citizens of a programming framework, thereby allowing to devise automated verification techniques able to detect where and why DoS may occur.

In particular, we start from the assumption that in a distributed system DoS reduces to absence of expected data, due to missing or corrupted communication, and consequential need for enforcing a default behaviour, a problem tackled by the introduction of the Quality Calculus. Then, we extend the original intuition behind the Quality Calculus and explore its implications in various scenarios, providing versions of the calculus for point-to-point and broadcast communication models, cryptographic reasoning, and cost considerations. Our Quality Calculi are complemented by a number of static analyses that exploit their peculiarities. The analyses we present are implemented as satisfiability problems, taking advantage of the efficiency and scalability of modern off-the-shelf solvers.

Obviously, full-resilience against DoS attacks cannot be achieved but combin-

ing the two views mentioned above, that is, looking at attacks directed against single components as well as to the impact of unavailability of components on the entire system. For this reason, we investigate a cost-aware version of the Quality Calculus and a corresponding static analysis able to estimate the cost to an attacker to interact with a system, so as to set bridges between our proposal and the cost-benefit analysis that informs actual countermeasures.

The ultimate goal of this dissertation is to support the following claim.

Language-based technologies offer a unifying approach to deal with the consequences of DoS, by means of a framework for facilitating the development of programs that follow a planned behaviour when expected information is unavailable. The modelling language can be supplemented by formal analyses enforcing such robust code to be produced.

The developments in this dissertation validate this claim as outlined in § 1.3 below.

1.3 Synopsis

In order to understand the organisation of this dissertation, a brief account of the chapters and of their connections is illustrated in the following.

Chapter 2 briefly surveys the three cardinal points of this dissertation – process calculi, static analysis, satisfiability – and explains how such weapons are combined in order to tackle the problem of unavailability.

Chapter 3 reviews existing literature on DoS, focusing in particular on formal attempts to give a foundational characterisation of unavailability, as opposed to practical solutions engineered to fight specific instances of the problem. Moreover, the chapter briefly presents CPSs and argues that they fall outside the range of existing methods for coping with DoS, and instead benefit from the defensive, availability-aware programming style developed in the subsequent chapters. Our discussion of CPSs is based on [Vig12].

Chapter 4 introduces the Quality Calculus, which we are to develop throughout the dissertation. We argue how the absence of expected communication can be explicitly addressed *already at the syntactic level*, thus allowing to specify a syntax-driven SAT-based static analysis to solve availability queries. The robustness analysis tackles DoS at the network level, where unavailability is caused by lack of expected communication. The chapter is based on [RNV12] and other work currently under submission.

Chapter 5 shows how the basic calculus can be lifted to reason about DoS caused by improper information, and lift the analysis to encompass the more complex task of pinpointing where absence of availability is due to the semantics of the communication, ideally addressing DoS at the application level of the Internet protocol suit (TCP/IP). The chapter is based on work under

submission.

Chapter 6 builds on Chs. 4,5, which delineate the paradigmatic features of availability, and instantiates those ideas to more concrete scenarios, presenting them in a broadcast communication model where cryptographic reasoning is implemented by means of equational theories; the broadcast model is to be exploited in the developments of Chs. 7,8. The chapter is based on [VNR13].

Chapter 7 explores how quantitative considerations enhance the developments of Chs. 4,5,6 so as to overcome theoretical qualitative answers in favour of more realistic estimate of the likelihood of DoS or of the cost of enforcing availability. The underlying intention is to reconcile the Quality Calculus approach and existing cost-based techniques discussed in Ch. 3. The chapter is based on [VNR14c].

Chapter 8 levers the intuition behind Ch. 7 to provide graphical representations of the attacks detected by the analysis (attack trees). The ultimate aim is to set bridges between the formal developments of previous chapters and the community of professionals that need appealing ways to convey security-related information. The work is based on [VNR14a].

Chapter 9 presents some concluding remarks and outlines open questions to be investigated in future work.

On the whole, Chs. 2 and 3 give the essential background to position this dissertation, while Chs. 4 to 8 contain the technical developments supporting our thesis.

Chapter 2

Setting the Scene

> Everything in computer science is a fixpoint. If something is not a fixpoint, then it is not worth studying.
>
> Flemming Nielson

This chapter aims at introducing the basics upon which the subsequent technical developments rest. Chapters 4,5,7,8 are heavily informed by a tripartite structure, according to which

1. a formal modelling language is introduced, able to describe a given class of systems (§ 2.1);

2. an abstract representation of such systems is produced, limited to capture some key behavioural traits (§ 2.2); and,

3. an automated reasoning technique is relied on to answer queries about such behaviours (§ 2.3).

These three elements take the form of a process calculus (1), a translation into logic formulae (2), and an application of SAT or SMT to such formulae (3).

Given a modelling language, the abstract representation and the reasoning technique define a static analysis on it in the style of flow logic. In Ch. 6 we shall explore a language supporting the subsequent developments, and therefore we deviate from this structure restricting to point (1).

2.1 Process Calculi

The phrase *process calculi* denotes a diverse family of formal languages for the description of concurrent systems. With "formal language" we mean a set of strings that obeys precise syntactic rules, often given in terms of a Context-Free Grammar [HMU06], and that are associated a rigorous semantics [RN07]. A popular perspective on process calculi suggests to look at them as a model of computation for *concurrency*, as much as the λ-calculus can be regarded as a model of computation for recursive functions.

One main difference between these two worlds is already suggested in the plural "calculi": while with the λ-calculus Church aimed at the fundamental structure of recursive computation, and thus at identifying the smallest set of primitives able to capture recursive functions, there exists a seizable number of process calculi. According to Milner [Ber05], this difference is justified if we construe a process calculus as a *modelling* language for concurrency. The emphasis on modelling triggers a state of tension between the quest for the foundations of concurrency ("the smallest set of primitives") and the capability of expressing real world behaviours in a natural manner, avoiding the intricacies of encoding in lower-level, artificial formalisms. Notwithstanding this, calculi such as CCS [Mil80] and the π-calculus [Mil99] have provided profound insights into the nature of concurrent computation. The two ends of the spectrum are summarised by Milner himself in his Turing Award lecture [Mil93b]:

> *I reject the idea that there can be a unique conceptual model, or one preferred formalism, for all aspects of something as large as concurrent computation [...]. We need many levels of explanation: many different languages, calculi, and theories for the different specialisms. The applications are various [...]. We surely do not expect the terms of discussion and analysis to be the same for all of these. But there is a complementary claim to make, and it is this: Computer scientists, as all scientists, seek a common framework in which to link and to organize many levels of explanation; moreover, this common framework must be semantic, since our explanations (including programs) are typically in formal language.*

As a matter of fact, since the seminal work of Milner and Hoare [Hoa85], over the last 40 years process calculi have proven useful languages for studying a great many real-world features that are crucially related to concurrency, including mobility, distribution, composition, discrete and real-time computation, stochastic interaction, and security. Moreover, orthogonally to the investigation of such foundational issues, process calculi have been applied to modelling a set of strikingly varied application domains well beyond software systems, including for instance the study of business organisations [Puh06] and biology [PPQ05, DB12].

Among the reasons that foster such a wide adoption of process calculi, we would like to stress their affinity to programming languages, the compositional

nature of process-algebraic models, and their mathematical elegance. Modelling a system in a process calculus is an experience similar to programming a piece of software exhibiting the behaviour of interest, thanks to the *operational* mind-set customary to these calculi. Moreover, such programming experience is essentially *compositional*, for the overall target model emerges from the interaction of smaller constituents according to well-defined composition rules.

As for the mathematical foundations, process calculi enjoy a largely-algebraic structure, to the point that the phrase "process algebra" is perceived as a synonym of "process calculus" and often used interchangeably (the chief example is the Algebra of Communicating Processes, by Bergstra and Klop [BK86]). [Bae05] contains an algebraic introduction to process calculi, and we refer the reader to this work also for a brief history of the discipline with a rich chronological account of the historically-relevant bibliography.

This dissertation embraces the modelling approach to the development of process calculi, as opposed to the study of the universality of calculi such as the π-calculus, which is know to be Turing complete. The former approach, besides its relevance in Milner's own words, has led to recent calculi such as COWS, SOCK, SCC, and CaSPiS (all surveyed in [CDP+11]) for understanding service-oriented computation, and has suggested several novel paradigms for dealing with increasingly-important notions, such as quality-of-service. As we will discuss below, *programming abstractions* are central to the cognitive process that leads to understand complex behaviours. The other approach focuses on mapping more specialised languages into the π-calculus or its generalisations: in the Psi-calculus framework [BJPV11], for example, insightful extensions of π can be formulated in a uniform manner, such as the applied π-calculus [AF01], the Spi-calculus [AG98], the fusion calculus [PV98], and counting.

2.1.1 Programming abstractions: a linguistic fascination

The principle of *linguistic relativity*, popular as the Sapir-Whorf hypothesis, claims that the structure of a language affects the conceptualisation of the world by its native speakers. In its strongest formulation, the principle suggests that cognitive categories are shaped after linguistic categories. While such form of *linguistic determinism* is generally agreed to be false, there is still much debate about weaker perspectives according to which *language influences thought* to some extent.

For the way we control computers is still chiefly linguistic, we are offered ground to test linguistic relativity in the realm of computer science. Eminent scientists have indeed provided inspiring arguments in this direction, though not referring explicitly to the Sapir-Whorf hypothesis. We should like to mention here Kenneth Iverson's Turing Award lecture [Ive80], "Notation as a Tool of Thought", as a general peroration on the centrality of language in science, and some works by Robin Milner, closer to the technical subject of this dissertation. Comparing the "levels of description" that one can exploit to understand con-

cepts in computing, such as routine, parametric procedure, etc., Milner found it natural to relate them to linguistic notions, such as parts of speech, metaphors, etc., to the point that

> *the best of these parts of speech and the best of these metaphors become accepted modes of thought; that is, they become concepts.* [Mil06]

In this respect it is crucial to underline how the linguistic relativity hypothesis focuses on the *syntactic categories* of a language rather than on its vocabulary, contrary to what seems to be commonly understood [Pul89]. This attention to syntactic categories matches a substantial line of research sponsored by Milner and concerned with the investigation of programming primitives that model given aspects of the real world seamlessly, as mentioned above.

Designing a language where the syntax helps the essential traits of a problem to emerge to the surface, invites designers and developers to realise the problem, to reason about it in a natural way, and to offer solutions that can be understood and communicated easily. As an example, one can try to explain encapsulation and inheritance out of the object-oriented paradigm, and perhaps even succeed, but the effort necessary to grasp the missing conceptual superstructure is likely to kill the most pedantic attempter (along the lines of saying that everything can be done in assembly).

When it comes to process calculi, we could provocatively attack the indiscriminate use of the π-calculus to modelling and analysing higher-level behavioural features as a practice that leads to modelling and analysing jumbles, thus loosing grip on the essence of the problem that one is facing without providing any practical advantage.

As for unavailability, the topic of this dissertation, we observe the increasing severity of the problem, and we notice that no programming language offers first-class constructs supporting DoS considerations. The linguistic relativity hypothesis just outlined not only supplies a fascinating conjecture connecting these facts, but also suggests a potential way through. Relying on the affinity of process calculi with programming languages, this work can holistically be understood as an attempt to capture the essence of availability through programming abstractions, which should facilitate devising a new generation of programming languages that compel developers to consider DoS hazards.

To conclude this digression, let us remark that it is not meant to justify the publication of yet more slight variations of existing calculi. We firmly believe that new primitives must be introduced with due parsimony and on well-documented ground, and we deem this the case of the Quality Calculus. Finally, we acknowledge that the linguistic relativity hypothesis is nothing more than a trenchant conjecture extraneous to exact science, but we hope that evoking such an imaginative comparison will prove stimulating and fruitful.

2.2 Reasoning on Abstract Representations

Besides facilitating and supporting our understanding of complex artefacts, formal models allow to state precisely the properties such objects are expected to enjoy, eventually enabling to devise tools for checking such expectations in an automated manner.

The Turing-complete expressive power of a great many process calculi mentioned in § 2.1 is not a mere adjunct, but rather a requirement imposed by the complexity of the scenarios they aim at capturing. This feature also suggests that in some cases the properties we would like to investigate cannot be decided in finite time and with finite memory. Even when such properties are decidable in theory, they might not be so in practice, some instances of the problem demanding for example an amount of time that exceeds a human life.

In order to reconcile our desiderata with theoretical and practical limits, different approaches have been developed which revolve around the concept of *abstraction*. One viable class of techniques relies on the idea of *simulating* a system to verify whether a given condition is met in all executions. Whenever the number of executions is infinite, however, sampling must replace exhaustive investigation, and thus *probabilistic* results are provided as opposed to formal qualitative assurances. In this sense we can think of simulation as an abstraction method. Among such techniques we can number *model checking* [BK08], which explores transition systems expressing the behaviour of programs in terms of transition between states. Whenever the state space is finite and has tractable size, exhaustive exploration is possible; in case the state space is infinite or intractable, statistical techniques are relied on [LD10].

Another class of techniques tackles the problem by producing a less constrained version of a given model, hence a more abstract object and ultimately simpler to reason about. The basic tenet is that a loss in precision often makes a property decidable in practice. *Static program analysis* [NRH99] techniques embrace this view and aim at providing *safe* approximations to the dynamic behaviour of a program by inspecting its syntactic structure, hence *statically*. Here "safe" means that the approximation computed by the analysis is a superset of the correct answer to the problem (possibly not too large a superset).

There are two classical approaches to the design of static analyses. *Semantics-directed* techniques *calculate* the analysis result from a semantic specification: this the approach of *abstract interpretation* [CC77]. *Semantics-based* techniques focus instead on specifying the analysis and rely on a posteriori validation: formal soundness and (sometimes) completeness theorems are formulated that allow understanding to which extent the results on the abstract model (analysis) carry to the original model of the system (semantics). Semantics-based approaches include type systems, monotone frameworks, and flow logic.

Among the arrows in the quiver of static analysis, we adopt a *flow logic* [RN02, RNP12] perspective, for it offers a clear separation of concerns between (*i*) the specification of the analysis (abstract domain), (*ii*) its soundness with respect to

the semantics of the language (correctness theorems), and (*iii*) the computation of the best analysis result (implementation). In Ch. 7, for instance, the analysis is specified in terms of an optimisation problem, which can be solved resorting to any of the available techniques, each suiting a particular context. Moreover, flow logic overcomes the rigid separation between data flow and control flow analyses, that does not fit process-algebraic models.

Our framework rests on static analysis developments as those of Chs. 4,5,7,8. In order to abstract some details of the original process-algebraic models, we resort to encodings into propositional or first-order logic, where the aspects of interest are reduced to sets of logic formulae (sometimes also referred to as *constraints*). The robustness analysis of Ch. 4, for instance, is concerned with what input variables are bound on the way to a program point, and thus all the information characterising output actions and creation of new objects is disregarded in the abstract domain. The basics of propositional and first-order logic are covered, e.g., in [Sch89].

The analyses we develop are to be understood as enforcement mechanisms guiding the programmers, as envisioned by Dijkstra [DDH72, § I.7]:

> *In my life I have seen many programming courses that were essentially like the usual kind of driving lessons, in which one is taught how to handle a car instead of how to use a car to reach one's destination.*

Borrowing this trenchant metaphor, we would say that a calculus prescribes "how to handle the car", that is, what systems are legal and what are not, whilst an analysis suggests meaningful routes to "reach one's destination", i.e., facilitates identifying systems that enjoy the property of interest, e.g., robustness against unreliable communication.

Once more, this point of view corroborates the usefulness of proposing a new calculus, although remaining in the range of the π-calculus. Promoting the paradigmatic behaviour of interest as a first class citizen enables to develop more precise analyses, and to interpret the outcome of the verification in terms of the same behavioural categories.

2.3 SAT and SMT

Once an abstraction of a process-algebraic model is obtained, we aim at establishing the properties of interest in an automated manner. Among automated reasoning methods for logic problems, the satisfiability approach is gaining increasing relevance thanks to the capability of modern solvers to cope with real-world instances and to their broad range of applications.

Propositional satisfiability (SAT) is the problem of determining whether there exists an assignment of Boolean values to the variables of a given propositional formula, such that the formula evaluates to true (tt) [MZ09]. Whenever

this is the case, the formula is said to be *satisfiable* (or *consistent*), otherwise the formula is said to be *unsatisfiable* (or *inconsistent*). Given a formula φ, we call *model* an assignment of values to its variables such that φ evaluates to tt, and we write $m \models \varphi$ (m *models* φ, m *satisfies* φ, or m *is* a model for φ). Otherwise, we write $m \not\models \varphi$ if the assignment m is not a model for φ. If a given formula is satisfied by every model it is said to be *valid*. If φ is valid, then $\neg\varphi$ is unsatisfiable. The problem can be generalized to non-Boolean logics.

As an example, consider the formula φ defined as

$$(x_1 \vee x_2) \wedge x_t \wedge (x_l \vee x_r) \wedge (\neg x_r) \wedge x_l$$

which we will encounter again in § 4.4, where $\wedge, \vee, \neg$ denote propositional conjunction, disjunction, and negation, respectively. The formula is satisfiable but not valid. An assignment m such that $m \models \varphi$ is

$$[x_1 \mapsto \mathsf{tt}; x_2 \mapsto \mathsf{ff}; x_t \mapsto \mathsf{tt}; x_l \mapsto \mathsf{tt}; x_r \mapsto \mathsf{ff}]$$

where m is here written as a map from the variables occurring in φ to truth values. Flipping x_1 to ff in the model m, for instance, we obtain an assignment that does not satisfy φ. For the sake of brevity, sometimes we shall adopt a functional notation and write $m(x) = \mathsf{tt}$ whenever m maps x to tt, and $m(x) = \mathsf{ff}$ otherwise. Moreover, it is worthwhile noticing that we call the propositional symbols (atomic formulae) "variables", in order to stress that we are somehow free to decide their values, but also the term "constant" is widely used.

The formula φ' defined as

$$(x_l \vee x_r) \wedge (\neg x_r) \wedge (\neg x_l)$$

is instead unsatisfiable, as x_r, x_l are required to be ff by the second and third conjuncts, respectively, entailing the impossibility of satisfying the first conjunct.

SAT was the first problem to be shown NP-complete [Coo71], and since then a substantial effort has been devoted to devise efficient SAT-solving techniques. In the last decade SAT gained a renovated attention, for the advancements in software and hardware technologies allow solvers to deal with real industrial problems, whose dimensions can scale up to millions of variables. Moreover, such improvements widened the scope of applications and thereby of research, enabling to consider problems whose constraints are interpreted in theories not limited to propositional logic.

Satisfiability Modulo Theory (SMT) extends the SAT problem considering a number of background theories (e.g., first-order logic, the theories of equality with uninterpreted function symbols, integers, real numbers, arrays), thus enriching the grammar of formulae and capturing more complex scenarios [dMB11]. SMT solving combines SAT solving with dedicated solvers for the theories. First, the atoms of a theory, e.g., linear inequality constraints over the reals, are mapped to fresh propositional variables. If the so-obtained SAT

abstraction of the original SMT problem is unsatisfiable, then the SMT problem is unsatisfiable, too. Otherwise, if there exists a SAT model m, the theory solver is used to check whether m is compatible with the theory, e.g., if those inequalities mapped to $\mathtt{tt}$ can indeed be satisfied in the domain of reals, hence obtaining a model for the SMT formula; if not, the SMT solver backtracks and a new SAT model is sought.

In the following, we shall focus on SMT formulae where the theories in question are first-order logic and the theory of Equality with Uninterpreted Functions (EUF). The latter is also known as the *empty theory* or the *free theory*, as it contains no equation other than those implied by equality being an equivalence relation and by the definition of function (identical pre-images are mapped to identical images, in this context also called "congruence property"). Consider the following SMT formula φ:

$$a \neq b \wedge f(a) = a \wedge f(a) = b$$

where a, b range over Boolean values and f maps Boolean to Boolean. While the SAT abstraction of φ is satisfiable, as all the conjuncts are mapped to fresh propositional variables, the formula is unsatisfiable in the theory of EUF, as the axiom that f behaves like a mathematical function is violated.

The SAT or SMT problem can be formulated in different ways. The *decidability* version focuses on the existence of a satisfying assignment, the *constructive* version computes it, and diverse *optimisation* versions rank models with respect to given parameters such as the number of clauses that can be satisfied (e.g., MAX-SAT) or the weight of satisfied clauses (e.g., Weighted MAX-SAT).

A number of SMT solvers has been proposed, developed both by universities and private companies, and they compete in periodic contests. Among those, we chose to build our framework on top of Z3 [dMB08, dMB], by Microsoft Research, as it supports the theories we need, its performance stands out in a number of categories, and because APIs are available for a handful of programming languages.

2.3.1 Theoretical complexity versus performance

In the last few years the feeling that SAT problems are feasible in practice has gained momentum, despite the unappealable complexity result. Such feeling is witnessed by the organisation of a panel on this subject at Vienna Summer of Logic 2014, featuring some of the most active experts in the field of automated reasoning. Commenting the question *Why SAT solving is working in practice much better than Cook's theorem would expect us to believe?*, Karem Sakallah reported some statistics drawn from recent SAT and SMT competitions that help shed light on the riddle. Facts and figures show that the largest instances of successfully-solved industrial SAT problems are of the order of 10 millions

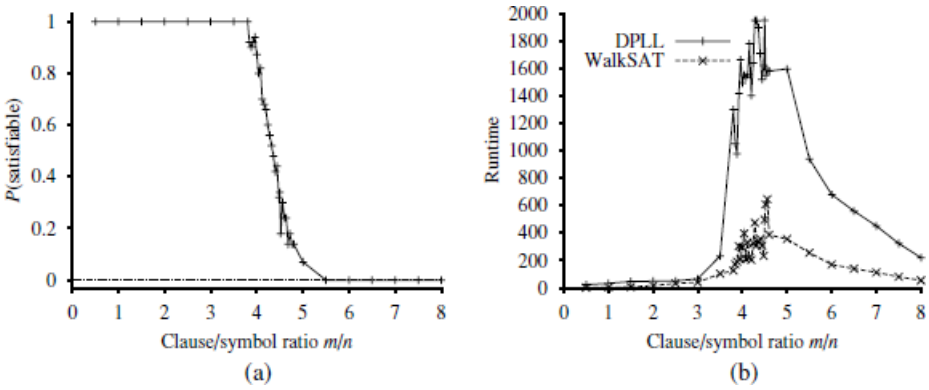


Figure 2.1: The clause/symbol ratio for $n = 50$ and its relationship to probability of satisfiability (a) and running time measured in number of recursive calls to DPLL (b) for random 3-CNF formulae, from [RN09].

variables, whereas in case of randomly generated problems the horizon is at about 50 thousand variables. Moreover, the size of solved industrial problems is increasing much faster than the size of solved random problems. Thus, it seems that "real" instances of SAT problems tend to enjoy a structure that can be efficiently exploited by solvers. Details on these considerations are discussed in [SMS11].

Earlier discussions of the same phenomenon are in [Lip09, Ch. 9,13] and in [RN09, § 7.6.3]. The latter, in particular, briefly reports on research on the so-called "satisfiability threshold conjecture", which relates the probability of *random* formulae to be satisfiable to the clause/symbol ratio m/n. For small values of m/n a problem is under-constrained, hence the probability of satisfiability is close to 1, as one would expect. Interestingly, it has been observed that such a probability drops sharply around $m/n = 4.3$, as summarised in Fig. 2.1. For recent work on the subject the curious reader may refer to [Ach09, COP13]. Even more interesting is to observe how the ratio m/n impacts performance: the running time of different SAT algorithms seems to follow a Gaussian-shaped distribution with mean at $m/n = 4.3$, for which the running time is the highest, while for smaller (under-constrained problems, likely satisfiable) and for greater values (over-constrained problems, likely unsatisfiable) the running time decreases.

Chapter 3

Denial-of-Service

Virgil Gligor: "Do you agree that in an open network fundamentally denial of service cannot be prevented, but that if the attack cost exceeds the gain from the attack it is unlikely that a denial of service attack would take place?"
Tuomas Aura: "Yes. That is precisely the goal here, that the cost of the attack should be greater than the damage caused."
Ross Anderson: "Presumably this does not apply to an attack for glory or revenge?"

[Aur01]

The dramatic changes undergone by the technological landscape in the last decade have led to a situation where almost every service, despite its criticality for individuals or the entire society, is deployed in an interconnected single infrastructure, making DoS a central issue. Ensuring availability is nowadays as important as guaranteeing confidentiality and authenticity, or even more, for it is meaningless to protect unavailable services. Nonetheless, while we are cleverer and cleverer at mastering confidentiality, integrity, and authenticity, our ability to counter availability attacks did not advance alike. Moreover, the cryptographic solutions that address some security properties cannot be relied on solely to achieve availability: one can provocatively argue that the more we instrument a system with cryptographic mechanisms, the more we leave it wide open to DoS attacks – substantiating the idea that DoS is a fundamental problem. A similar conclusion is drawn by Gollmann [Gol11, § 1.1.5], who comments

upon flooding attacks as follows: "There is a distinctive lack of security mechanisms for handling this problem. As a matter of fact, too restrictive security mechanisms can themselves lead to denial of service".

In this chapter, we shall review some definitions of availability, present some practical traits of DoS attacks together with the countermeasures that have been proposed, discuss formal attempts to characterising the problem, and finally look into the new challenges arising in the realm of CPSs.

3.1 A Bird's Eye View

Ideally, availability concerns span every branch of engineering, building dependable computer systems being just one such branch. Before turning our attention to computer science, it is worthwhile reviewing briefly the traditional engineering perspective on the subject. For a thorough treatment of the topic the reader is referred to [KK07].

3.1.1 Availability in theory

Traditionally, in engineering disciplines, availability is just one of the metrics that characterise the broader notion of *dependability*, often a synonym for *fault-tolerance*. As such, availability enjoys a precise mathematical definition: *the fraction of time that a system is operational over a fixed interval.* This definition is put in contrast with that of *reliability*, characterised as the *probability* that a system has worked *continuously* in a fixed interval. Hence, reliability and availability fit different purposes: the former is suitable for applications where continuous operation is crucial, while the latter is appropriate for applications where momentary disruption can be tolerated, average behaviour being what matters. In this sense, an on-line booking system is concerned with high availability, for brief interruptions are not catastrophic, whereas we would require an aircraft to enjoy high reliability!

A neighbouring concept is that of *graceful degradation*, the capability of a system to continue operating even in the event of failure of some of its components, perhaps reducing the expected quality of the outcome that it is supposed to yield. Clearly, any form of degradation can be made systematically graceful only if great care is exercised at design-time, so that graceful lower-quality alternative plans are provided for. As we shall see in Ch. 4, this idea is the conceptual cornerstone upon which the Quality Calculus rests.

As for computer science, the concepts of reliability and availability have a direct counter-part in the study of computer networks. The notions of node and line *connectivity*, used to describe the number of nodes or edges that needs be taken down to disconnect two sub-networks, respectively, corresponds to reliability. Connectivity only distinguishes whether the network is connected or disconnected, as reliability is mainly concerned with the occurrence of a failure

in a given time interval. A pool of measures for estimating the *robustness* of a network corresponds instead to availability, such as average or maximum distance between nodes.

The network example, however, is still chiefly in the realm of traditional engineering. When it comes to information technology, and security in particular, availability takes on a different and somewhat less precise, broader meaning. The International Standard ISO 7498-2 [ISOb], which defines the ISO/OSI architecture for communication security, states the following

Definition 3.1 (ISO/OSI Availability) Availability is the property of being accessible and usable upon demand by an authorised entity.

Unsurprisingly, DoS is then defined as *"the prevention of authorised access to resources or the delaying of time-critical operation"*. Similarly, the Common Criteria for Information Technology Security Evaluation [ISOa], which superseded a number of former standards such as CTCPEC, ITSEC and TCSEC (better known as the *Orange Book*), relates the notion of availability to the one of authorisation, and also mentions the relevance of availability metrics, without specifying any.

It is crucial to note that both standards refer to authorisation. On the one hand, it is quite obvious that the unavailability of a service to a party not entitled to use it does not amount to DoS. On the other hand, authorisation suggests authentication, which is usually implemented in terms of cryptography, and we have already put forward that this may be a hook for DoS attacks, cryptographic operations being in general computationally expensive. Already at this high level of discussion we appreciate how challenging and intriguing a problem we face.

3.1.2 Denial-of-Service in practice

There exists a rich corpus of taxonomies for the categorisation of DoS attacks and countermeasures; we refer the reader to [RC11] for a brief survey of proposed nomenclatures. Nevertheless, moving from theory to practice, it is at first surprising how narrow the picture becomes when considering everyday applications of computer security to DoS. Here unavailability becomes often a synonym for network-flooding attacks: the generation of an amount of traffic that exceeds the capacity of the target server, which consequently cannot serve legal requests for a given period of time. This is an instance of *resource-exhaustion* attack, where the resources in question are the server memory and bandwidth.

Even though network flooding is just one potential source of DoS, it is evident how in the last decade this sort of attacks has occurred increasingly and caused increasing damage, to the point that major attacks have been reported even by mainstream media. This trend culminated in the distributed DoS (DDoS – carried out by a number of agents) campaigns coordinated by the activists

collectively known as Anonymous and related to the ongoing political and economic turmoil in Western countries (e.g., the WikiLeaks and Occupy Wall Street cases). Anonymous's activities led to defacing web-sites belonging to major corporations such as PayPal, Visa, MasterCard and to government bodies of various countries (e.g., those involved in the so-called Arab Spring). The press coverage is too vast for being systematically cited. Selected references include [Som, Lav] (single cases) and the rich time-line of events associated with Anonymous featured on Wikipedia [Wikb] and the references provided therein.

The spread of network-based DDoS attacks is greatly facilitated by the existence of automated tools [GJM12] that allow flooding a target address with communication packets (TCP, UDP, ICMP, or a combination of them) requiring almost no technical skill to the attacker: Low Orbit Ion Cannon-like tools [Wika] run on a variety of operating systems and come with an intuitive graphical user interface.

Existing techniques for countering DoS, briefly surveyed in the following section, mainly focus on the network segment. On the contrary, little attention is paid to attacks exploiting the semantics of the communication or physically tampering with the target. Whilst we acknowledge the importance of improving our capability of actively defending systems against network-level attacks, at the same time the successfulness of great many DDoS attacks raises the question whether there exists a systematic approach to designing systems that degrade gracefully when some components become unavailable. Besides addressing the problem of successful flooding attacks, the idea to cope with the consequences of DoS tackles the diverse sources of unavailability in a uniform manner.

3.1.3 Countermeasures

Countermeasures to DoS can be categorised into proactive and reactive methods. As it is often the case, the best resilience guarantees cannot be obtained but combining the two approaches.

Reactive techniques are *dynamic*, in the sense that they try to *detect* ongoing attacks and react upon detection. These methods, mainly developed for network-level attacks, aim at telling legitimate from malicious traffic by analysing patterns (e.g., [IB02, YPS04]). Their main shortcoming consists in that an attack may target the detection process itself, thus compromising the defence architecture. On the formal methods side, statistical model checking [LD10] has been used to verify quantitative properties of some typical defence patterns formally modelled in Maude [AGG+05, EMA+12]. Categorisation techniques typical of information retrieval applications have been used to learn how legitimate traffic looks like [KAv14].

Proactive approaches are *static*, in the sense that they try to *prevent* attacks from taking place, and are usually based on cost-benefit considerations over attacks, as highlighted in the vibrant discussion quoted at the beginning of the chapter. For the formal considerations of § 3.2.2 are shaped around these

techniques, it is worthwhile briefly reviewing some applications that fall in this group.

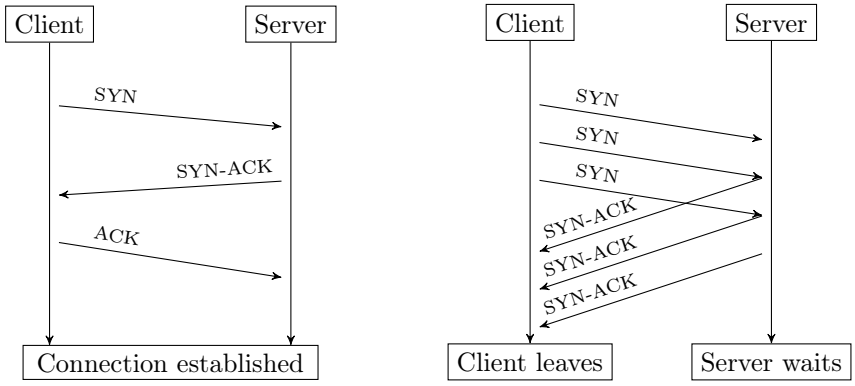
SYN cookies. One of the first examples of a proactive cost-based technique is given by *SYN cookies* [Ber96], available for instance in the implementation of TCP/IP shipped with Linux (but not enabled by default).

In the three-way handshake mode of TCP (Fig. 3.1(a)), with *SYN flooding* (Fig. 3.1(b)) we refer to an attack in which the attacker(s) floods a target TCP server with connection requests (known as SYN packets in TCP), without engaging in the protocol any further. In this way, a number of connections is hanging on the server side, which has to keep resources allocated for some time in case the client replies. As the server resources are limited, at some point the server will stop accepting new requests, until some of the old ones expire, hence becoming unavailable over a given time frame. This is perhaps the widest-known flooding attack, and is documented since 1996 [Edd06].

The idea behind SYN cookies reduces to exploit the TCP sequence numbers, used by the server to rebuild the packet stream, as light-weight authenticators. These smart sequence numbers are such that the server does not need to record the connection request (SYN packet) of a client: it instead replies with a SYN-ACK packet whose sequence number n is a function of the server and client IPs and ports as well as of a timestamp. The client is then expected to reply with a packet (ACK) whose sequence number is $n + 1$, so that the server can check its validity. The key-point is that pure SYN flooding is ineffective, for the server does not store any information about incoming requests, and therefore its memory is unaffected by such an attack.

SYN cookies offer a great example of a quantitative answer to DoS hazards, both for attackers and defenders. On the attacker side, before being able to forge a valid connection and flood the server, a malicious agent must guess a valid sequence number, and this is considered to be difficult due to the use of cryptography (the sequence number is hashed) and to the requirement of freshness (the timestamp is incremented every 64 seconds). On the defender side, there is a price to be paid in terms of efficiency, as the server has to spend some CPU time checking the sequence number of the client reply, but such time is assumed to be negligible.

Client puzzles. The root of most unavailability resource-exhaustion attacks lies in the disparity between the server and the client load. Consider TCP: SYN flooding is possible because the server has to commit some of its resources without the client sharing the burden. In order to address this incongruity, some techniques require the client to solve a computational expensive problem before the server accepts a service request. Such methods, collectively referred to as *client puzzles*, have been proposed both for TCP/IP [JB99] and for higher-level protocols [ANL01, WJHF04]. At the highest level of the application stack,



(a) TCP three-way handshake. (b) SYN flooding.

Figure 3.1: TCP three-way handshake and SYN flooding.

CAPTCHA tests (Completely Automated Public Turing test to tell Computers and Humans Apart) are a now-common technique to fight DoS (as well as other threats). Game-based analysis has been proposed as an effective formal verification technique for client puzzles [MS05].

Availability by Design. Proactive methods include the development of software which is robust by design against unavailability attacks according to qualitative considerations as opposed to cost-based reasoning. The same view of the Quality Calculus about planning alternative behaviours is applied to engineering a network-level defence strategy in [KPYK13], where the system enters different states depending on the traffic load. Differently from the Quality Calculus, that work proposes an applicative solution rather than a principled approach to software design.

Unavailability due to the semantics of the communication is investigated in [CJI+09], where several static analysis techniques are combined to detect inputs that activate costly executions. Still, the focus is on resource exhaustion caused by algorithmic complexity, while our notion of semantic DoS targets the application level and coincides with Gligor's concept of misbehaved service (cf. § 3.2.1).

It is worth observing how in the Quality Calculus acceptation availability by design can be construed as being both reactive and proactive. At the component level we are reactive: *if* expected information is not arriving, thus jeopardising the ideal behaviour, *then* we enforce an alternative plan – only when we *detect* that some other component is unavailable, for it is not responding in due time. At the system level we are instead proactive: each component being able to perform its duty even in the absence of expected information, we

prevent the overall system from entering an unavailability state. Again, notice how the focus is shifted from reactivity/proactivity with respect to attacks to reactivity/proactivity with respect to unavailability in general.

3.2 Formal Approaches to DoS

We have briefly argued how existing work on DoS is skewed towards network-level vulnerabilities. This restriction allowed to develop practical solutions to specific cases, but their proliferation hampered the isolation of general principles applicable to the problem per se. In the following, we shall turn our attention to the efforts put in characterising the nature of DoS, irrespective of the source or of the technological carrier of an attack.

3.2.1 Early steps

Virgil Gligor is usually credited with the first attempt to give a foundational and systematic treatment of DoS in computer science [Gli83, Gli84, Gli86].

According to Gligor, DoS is a *security problem* because it can result in unauthorised disclosure of information (e.g., the operational status of a sub-system). Moreover, integrity problems are a potential source of DoS, for it is not always possible to operate on compromised data. Therefore, DoS is a *distinct security problem* for it is related to two problems usually regarded as distinct ones, namely, secrecy and integrity. Finally, "if one defines as 'fundamental' a problem which remains when the cost of technology decreases to zero, then DoS is a *fundamental problem*" [Gli86, § 2].

However, according to Gligor, DoS is of secondary importance with respect to other security properties, for some instances of the problem (though not all) can be solved by ensuring integrity and authenticity. Whilst this was possibly reasonable in 1986, it is not any more, for the reasons stated in the introduction to the chapter.

This being said, Gligor attempts to formalise availability as *guaranteed access*:

Definition 3.2 (Guaranteed access) No authorised user is able to deny the access of any other authorised user to a shared service.

The key entities are thus a *shared service* and some *authorised users* entitled to accessing such service. It is implicit in the problem that the service is characterised by a *Maximum Waiting Time* (MWT) an authorised user has to wait before being granted access to the service: if no finite time bound exist, then DoS cannot take place, because the service is not promised to any user. If MWT is 0, then again DoS cannot take place, as the service becomes private (or local).

Definition 3.3 (Denial-of-Service) A group of authorised users G of a given service S is said to deny S to another group G' of authorised users if G makes S unavailable to G' for a period of time which exceeds the intended MWT of S.

This formulation is said to encompass all the definitions proposed in the literature until then, as surveyed in [Gli83, App. B]. The first implication of Def. 3.3 is that DoS can in some cases be eliminated by merely increasing the MWT. Second, it emphasises that DoS occurs when S is made *unavailabile*: Gligor observes that this happens not only when S is *unreachable*, but also when S responds within the MWT but does not behave *as expected*. DoS can indeed take place at the transport layer (S unreachable), or at the application layer (S misbehaved) of the Internet protocol suit stack. It is worthwhile observing that DoS at one level can be caused by actions occurring on other levels, e.g., physical destruction of the machine hosting S makes it unreachable, or selectively jamming a wireless signal can make S misbehaved.

Notice that the definition considers *authorised* users accessing and preventing access to S, while DoS occurs even if it is an unauthorised party who inhibits accessing S to authorised users (as in *malicious* flooding attacks). Gligor [Gli84] acknowledges that Anderson [And72] considers the possibility of an intruder causing DoS, but assumes that standard protection mechanisms can be used to rule out this threat. However, this seems to conflict with the intuition that cryptography can introduce new sources of DoS. Later in [Gli86], where the peculiarities of unavailability in networks are explored, the importance of malicious DoS gains more centrality, based on considerations that recall the coeval intruder model proposed by Dolev and Yao [DY83]. Most interestingly, Gligor concludes that *"most intruder-based DoS attacks cannot be prevented"*, but only be detected.

Levering these definitions and detailed case studies, *undesirable inter-user dependencies*, according to which a group of users becomes dependent upon the behaviour of another group of users, are identified as common denominator of DoS. Without analysing the conditions in detail, let us just note that Gligor himself acknowledges that such dependencies have no uniform nature or cause, even if a relationship to inadequate service-sharing policies and mechanisms can be tracked.

In later work, Yu and Gligor [YG90] tackle the problem of formally specifying user agreements, addressing those instances of DoS that occur even in absence of failures and of integrity violations. They identify unavailability as a property with both *liveness* and *safety* features. When DoS is caused by some users preventing other users from making progress it is construed as a liveness problem, whereas when some users cause other users to receive incorrect service it is construed as a safety problem. Most interestingly, we find again the distinction between what above has been characterised as network-level and application-level DoS.

One of the main contributions of the work of Yu and Gligor is the conceptualisation of a *resource allocator* as the nodal component to prevent unavailability, whose properties and policies are stated with temporal formulae. Subsequent work by Millen (cf. [Mil93a] and other versions) elaborates on the subject presenting more refined models for resource allocators, substantially agreeing with the definitions of [Gli83, Gli84, Gli86].

3.2.2 From qualitative to quantitative considerations

The early work surveyed so far suffers from a qualitative approach to availability, despite the awareness of mathematical characterisations of quantities strongly related to DoS. Nonetheless, Gligor had already foreseen the difficulty of preventing DoS *attacks*, comment which directly leads wondering *to which extent* unavailability can be prevented in a given system. This perspective, together with a more modern protocol-oriented mind-set, heavily informs Meadows's work on DoS [Mea99, Mea01], which remained a touchstone for most of the subsequent publications on the subject (in spite of the partial oblivion that affects earlier work).

Meadows starts out observing that there exists a number of practical approaches that fight DoS on the basis of cost considerations (cf. § 3.1.3), "cost" being very generally construed as any relevant quantity for the involved players. One option is to reduce the cost to the victim of engaging in a protocol and/or to increase the cost to the attacker; another is to increase the resources to the victim; a third is trying identifying the source of the attack (a particular case in which the cost to the attacker is identity disclosure). Whilst a naive approach to the latter may introduce new DoS risks, some cost-effective solutions have been proposed, such as combining weak authentication when a protocol initiates to strong authentication as it completes [SK97, AN04].

In a sense, Meadows's work offers a unifying formal framework where the different drivers of these techniques fit naturally. "Besides, it has all the charm of inventing the science of navigation while already on board ship", as Robin Milner happened to say about laying down logical foundations of informatics [Mil06].

In a nutshell, Meadows's framework aims at characterising unavailability as a cost-determined condition. In a realistic execution of a communication protocol, there is a cost attached to processing each message, both for legitimate participants and for attackers. Such costs include for instance generating nonces, storing received information, performing equality checks and cryptographic computations. Moreover, there is a budget that limits the resources available to each player. Hence, DoS occurs when the resources of a player are exhausted. A tolerance relation for a system can thus be defined in terms of pairs (c_s, c_a), where the attacker spends at most c_a for having the system wasting at most c_s cost units. All interactions yielding costs that exceed such threshold are potential source of DoS.

The central technical developments sustaining the main idea consists in an

annotated protocol narration language, and in an elegant definition of the cost structure characterising the players' operations. In particular, a map from actions to costs is required where costs are partially ordered elements of a monoid, thus allowing full generality in describing the resources available to the players (including developing distinct cost sets for the defender and the attacker and time-dependency via non-commutative monoids). We shall build on this idea in Chs. 7,8.

A message exchange in Meadows's annotated narrations has the following form:

$$A \longrightarrow B : \underbrace{T_1, \ldots, T_k}_{\mathsf{cost}(A)=\bigoplus_i \mathsf{cost}(T_i)} \quad ||M|| \quad \underbrace{O_1, \ldots, O_n}_{\mathsf{cost}(B)=\bigoplus_j \mathsf{cost}(O_j)}$$

where the T_i's are the actions principal A must perform before sending the message M and the O_j's are the actions principal B must perform upon receiving the message, and $\oplus$ is the monoid operator used to combine costs. For instance, sending the message $M = \alpha^{X_A}$ requires exponentiating and storing terms on A's side, and storing the message on B's side. Interestingly, in the case of SYN flooding a malicious initiator A needs to perform no pre-action T, while the server B has to store the incoming request and thus performs at least one O: the notation gives an immediate and formal account of the issue behind the problem, i.e., load unbalance. In general, costs determine the presence of unavailability threats: if $\mathsf{cost}(A) \ll \mathsf{cost}(B)$, then the protocol is potentially subject to resource-exhaustion attacks.

The generality of the cost structure encompasses a number of notions of costs, from numeric ones (e.g., money, energy, time) to symbolic ones (e.g., "low" or "high" risk of post-action forensics), and thus lends itself to describe the ever-growing taxonomy of DoS attacks and countermeasures.

Meadows's cost-based framework inspired a number of theoretical and practical developments. Among the former, Lafrance and Mullins [LM03] develop a bisimulation-like interference-based method to detect DoS. The idea is to characterise a DoS-robust system as a program that does not change his behaviour when running in a hostile environment: in particular, if the presence of an adversary triggers new costly traces, then the system may be subject to DoS attacks. Pilegaard et al. [PHS03] combine Paulson's inductive approach to protocol verification with Meadows's framework, in order to automate the computation of the analysis. On the practical side, Meadows's framework has been used to study protocols such as IKE [Ram02] and JFK [SGNB06].

3.3 DoS in Cyber-Physical Systems

Before delving into the presentation of the Quality Calculus, let us briefly review the class of systems that inspired its development, i.e., Cyber-Physical Systems (CPSs). In particular, we shall concentrate on the security-related features of

such systems, and show how their vulnerability to DoS advocates reconciling graceful degradation and cost-based reasoning.

CPSs are complex systems that monitor physical processes by means of interconnected networks of sensors, whose measures are exploited for acting on the sensed environment in order to optimise an operational goal [Lee08, XRK08, SWYS11]. Such systems can be logically organised in two coupled layers:

- a *physical* layer, consisting of sensors and actuators that interact with given facets of the environment (e.g., physical parameters like temperature, humidity, etc.),
- and a *cyber* layer, in charge of transforming the sensed data into information, to be exploited for driving the environment toward a given goal, possibly by means of the actuators.

Typically CPSs consist of a significant number of devices (up to tens of thousands of nodes), each one capable of sensing and/or actuating, computing, and transmitting data. Such systems are increasingly exploited in the realization of critical infrastructures (e.g., power grid, healthcare, traffic control, defence) as well as general-purpose personal applications (e.g., home automation, entertainment) [STCE14].

A fruitful starting point for discussing the security of CPSs is to observe that both the physical and the cyber layer are subject to threats, and that an attack on one level may compromise the operation of the other level. On the cyber layer all classic communication-based attacks are viable. On the other hand, physical attacks include destroying a component, removing it temporarily from the network, tampering with it in order to read or modify the storage or reprogram the control software. Whenever components are powered by batteries, communication-based attacks impact their life-span determining a cross-layer effect. Similarly, accessing the storage of a node grants an attacker the knowledge of its cryptographic identity (if any, including session keys), and therefore allows to take full control of the component also on the cyber layer.

We have shown in [Vig12] how such a cyber-physical attacker encompasses and generalises a great many attacker models used in protocol verification. Basin and Cremers [BC10] already observed that it is unclear whether any protocol would be correct with respect to an adversary who can get all the cryptographic material of the participants. From a verification point of view, we are thus fostered moving from a qualitative perspective to a quantitative perspective, as it does not make sense to wonder whether or not a system would be secure in face of such an attacker, but only *to which extent* the system is protected, or what is the *cost* an attacker incurs to achieve a given goal.

Orthogonal to the question whether an attack targets cyber or physical traits of a system, is the question whether unavailability is caused by an attack at all. There are at least other two sources of DoS: misfortune and design. A typical example of misfortune are unfavourable environmental conditions. As a matter

of fact, some applications require to deploy components in a vast geographical region, where they cannot be physically protected, neither from attackers nor from nature. By design we mean instead that unavailability may be a planned stage in the life-cycle of a node, with the aim, for instance, to optimise battery consumption.

Hence, CPSs push for reconciling graceful degradation and quantitative techniques in a uniform framework. On the one hand, unavailability may be a legal status of some components in a given time-frame, on the other hand the ways for achieving malicious DoS are plentiful. Full resilience against DoS cannot be achieved without being aware of the former and countering the latter.

3.4 Ready Set Sail

The overview of unavailability attacks and countermeasures we have presented, though brief, establishes some useful categories to position the developments of Chs. 4,5,6,7.

First of all, all the definitions of availability that we have reported include a demand for authorisation/identification. This seems to go hand in hand with the *reactive* approach to countering DoS, for it requires to tell legitimate from malicious operations, e.g., traffic analysis. Furthermore, different components may rely on different authorisation databases, suggesting that those definitions are well-suited for a *self-centred* perspective, in which the component takes care of its own availability. We shall instead work in *distribution-aware* settings and address unavailability as a system concern, allowing single components to become unavailable and working for graceful degradation.

What is more, dealing with the consequences of unavailability is agnostic with respect to the source of DoS (malicious, planned, inadvertent) and cope with resource-exhaustion, semantic, and physical attacks uniformly.

For presentation purposes, we shall follow the phylogenesis of the literature, presenting our distribution-aware approach to DoS first in a qualitative framework – both for network (Ch. 4) and application-level (Ch. 5) unavailability – and then extending it to encompass quantitative considerations (Ch. 7), with the ultimate aim of reconciling graceful degradation and cost-based approaches.

Chapter 4

The Quality Calculus: Modelling Availability

> It typically takes at least 10 to 20 years for a good idea to move from initial research to final widespread practice.
>
> [OGKW08]

Absence of communication is one chief consequence of the unavailability of communicating components in a distributed system. As we have argued in Ch. 3, focusing on the *effects* of unavailability accounts for coping with all sources of DoS, including faulty components or communication medium, unfavourable environmental conditions causing physical disruption, and active attackers. At system level, one of the most prominent effects of DoS is the absence of expected data due to missing communication. Hence, a main challenge in the development of distributed systems is to ensure that the distributed components continue to behave in a reasonable manner even when communication becomes unreliable, thereby stopping the domino effect induced by denying service to other components.

Computer science techniques can help ensure that software systems are hardened against the unreliability of communication. This calls for programming software components of distributed systems in such a way that a default behaviour is enacted when the ideal behaviour is denied due to the absence of expected communication. To this end, in this chapter we develop

- a process calculus, the Quality Calculus, for programming software components and their interaction, natively equipped with the notion of absence of communication, and
- a SAT-based analysis to determine the vulnerability of processes against unreliable communication.

The Quality Calculus is developed in §§ 4.1,4.2 and clearly inherits from calculi such as CCS and the π-calculus. Its main novelty consists in coupling non-blocking input binders with option data type, exploited to distinguish syntactic elements that carry actual data from elements that carry potential (optional) data. In order to fully exploit the capability of progressing without all the expected information, that is, without all the prescribed input to be satisfied, *quality binders* are used that range over a number of sub-binders, and can proceed when a given combination of them is satisfied. Finally, the language obliges to check whether or not an input variable carries actual data, allowing to act accordingly in the continuation.

The expressiveness of the Quality Calculus is considered in § 4.3 and an example in the context of a wireless sensor network is presented in § 4.4.

The SAT-based *robustness analysis* is developed in § 4.6. It is based on the view that processes must be coded in such a way that error configurations are not reached due to unreliable communication; rather, default data should be substituted for expected data in order to provide meaningful behaviour in all circumstances. Of course, this is not a panacea – default data is not as useful as the correct data, but often better quality-of-service can be obtained when basing decisions on default or old data, rather than simply stopping in an error state. As an example, if a braking system does not get information about the spinning of the wheels from the ABS system, it should not simply stop braking, rather it should continue to brake – perhaps at reduced effect to avoid blocking the wheels.

The analysis attaches propositional formulae to all points of interest in the processes; such formulae characterise the combinations of optional data that could be missing. This is useful for showing that certain error configurations cannot be reached; indeed, if a propositional formula is unsatisfiable, then the corresponding program point cannot be reached. The availability of extremely efficient SAT-solvers makes this a very precise analysis method with excellent scalability.

We present two equivalent semantics for the calculus, a standard reduction semantics (§ 4.2) and a semantics with explicit substitutions (§ 4.5). While the elegance of the former is superior for presentation purposes, the latter is necessary to formally prove the correctness of the robustness analysis. Moreover, the latter defines a novel way of integrating explicit substitutions in a process calculus, promoting the treatment of the environment to a *directed* set of structural rules. The equivalence of the two semantics (§ A.2) supports the claim that our approach to explicit substitutions can be seamlessly inherited by other

π-like calculi.

This chapter is mainly based on [RNV12], where the Quality Calculus has been first introduced. The explicit substitution semantics of § 4.5 and the formal proof of § 4.6.3 are contained in work currently under submission.

4.1 The Quality Calculus

A *system* consists of a number of process definitions and a main process:

$$\begin{array}{ll} \mathsf{define} & A_1(x_1) \triangleq P_1 \\ & \vdots \\ & A_n(x_n) \triangleq P_n \\ \mathsf{in} & P_* \end{array}$$

Here A_i is the name of a process, x_i is its formal parameter, P_i is its body and P_* is the main process. The syntax of processes is given in Table 4.1. A *process* can have the form $(\nu c)\, P$ introducing a new constant c and its scope P, it can be the parallel composition $P_1|P_2$ of two processes P_1 and P_2, and it can be the terminated process, denoted $\mathsf{0}$. An input process is written $b.P$, where b is a binder specifying the inputs to be performed before continuing with P. An output process has the form $t_1!t_2.P$, specifying that value t_2 should be communicated over channel t_1. $A(e)$ is the recursive call to one of the processes defined in the system, e being the actual parameter of the call. Finally, a process can be a case construct whose explanation we defer to later. In the following, we shall feel free to dispense with trailing occurrences of the process $\mathsf{0}$.

The main novelty of the calculus is the *binder* b specifying the inputs to be performed before continuing. In the simplest case it is an input guard $t?x$ describing that some value should be received over the channel t and bound to variable x. Increasing in complexity, we may have binders of the form $\&_q(t_1?x_1, \cdots, t_n?x_n)$, indicating that n inputs are *simultaneously* active, the *quality predicate* q determining when sufficient inputs have been received to continue. For the sake of readability, we shall use some abbreviations: q can be $\exists$ meaning that one input is required, or it can be $\forall$ meaning that all inputs are required; these and other examples are summarised in Table 4.5. More complex cases arise when binders are nested, as in $\&_\forall(t_0?x_0, \&_\exists(t_1?x_1, t_2?x_2))$ that describes that inputs must be received over t_0 as well as one of t_1 or t_2 (or both). If we assume that our quality predicates can express all combinations of arguments, then nested binders can always be linearised without changing the overall semantics; as an example, $\&_\forall(t_0?x_0, \&_\exists(t_1?x_1, t_2?x_2))$ has the same effect as $\&_q(t_0?x_0, t_1?x_1, t_2?x_2)$ if $q(r_0, r_1, r_2)$ amounts to $r_0 \wedge (r_1 \vee r_2)$.

As a consequence, when continuing with the process P in $b.P$ some variables might not have obtained proper values, as the corresponding inputs might have

Table 4.1: The syntax of the Quality Calculus.

$$
\begin{array}{rcl}
P & ::= & (\nu c)\, P \mid P_1 \mid P_2 \mid 0 \mid b.P \mid t_1!t_2.P \mid A(e) \\
 & \mid & \mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y) : P_1\ \mathsf{else}\ P_2 \\
b & ::= & t?x \mid \&_q(b_1, \cdots, b_n) \\
t & ::= & y \mid c \mid g(t_1, \cdots, t_n) \\
e & ::= & x \mid \mathsf{some}(t) \mid \mathsf{none} \mid f(e_1, \cdots, e_n)
\end{array}
$$

not been performed. In order to distinguish between successful and unsuccessful inputs we resort to a distinction between *data* and *optional data*, inspired by option data types in programming languages like Standard ML [MTHM97]. In the syntax we use terms t to denote data and expressions e to denote optional data; in particular, the expression $\mathsf{some}(t)$ signals the presence of some data t and none the absence of data. Returning to processes, the construct $\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y) : P_1\ \mathsf{else}\ P_2$ will test whether e evaluates to some data and if so, bind it to y and continue with P_1 and otherwise continue with P_2. It is worthwhile observing that since input channels and output range on terms (data), the calculus syntactically obliges to inspect the content of an input variable before using it. In this sense, at any point of the computation we know what information is certainly available and what is not, and the analysis shall exploit such syntactic perks.

The distinction between data and optional data copes with the potential unreliability of communication, allowing to proceed even when expected information is not arriving, perhaps using default values in else branches and thus degrading the *quality* of the outcome yielded by the process, hence the name of the calculus. As we have already put forward in Ch. 3, the framework defines a principled approach to graceful degradation.

In order to insist on the syntactic categories of terms/data and expressions/optional data, in the following we shall use the sets $\mathcal{X}$ of variables x that stand for expressions, $\mathcal{Y}$ of variables y that stand for terms, $\mathcal{C}$ of names c that stand for constant data, $\mathcal{O}$ of constant optional data o.

Clearly, more elaborate choices of syntax for expressions and terms are possible, including the possibility of distinguishing between them using a type system. However, for the sake of simplicity we have opted for two syntactic categories and therefore we also distinguish between functions g returning data values and functions f returning optional data values. We assume that such functions are total. In Ch. 6 we shall instead rely on a single syntactic category and distinguish between data and optional data by means of a simple type system.

A case construct for expressions similar to the one for processes, defined in the calculus of [RNV12], is omitted for the sake of simplicity for it does not

Table 4.2: The structural congruence $\equiv$ of the Quality Calculus.

$$P \equiv P \text{ (Ref)} \qquad P \,|\, 0 \equiv P \text{ (Nil)} \qquad P_1 \,|\, P_2 \equiv P_2 \,|\, P_1 \text{ (Com)}$$

$$P_1 \,|\, (P_2 \,|\, P_3) \equiv (P_1 \,|\, P_2) \,|\, P_3 \text{ (Ass)}$$

$$(\nu c_1)\,(\nu c_2)\,P \equiv (\nu c_2)\,(\nu c_1)\,P \text{ (New1)}$$

$$(\nu c)\,P \equiv P \text{ if } c \notin \mathsf{fc}(P) \text{ (New2)}$$

$$(\nu c)\,(P_1 \mid P_2) \equiv ((\nu c)\,P_1) \mid P_2 \text{ if } c \notin \mathsf{fc}(P_2) \text{ (New3)}$$

$$\frac{P_1 \equiv P_2}{P_2 \equiv P_1} \text{ (Sym)} \qquad \frac{P_1 \equiv P_2 \qquad P_2 \equiv P_3}{P_1 \equiv P_3} \text{ (Tra)} \qquad \frac{P_1 \equiv P_2}{C[P_1] \equiv C[P_2]} \text{ (Cnt)}$$

increase the expressive power of the calculus.

We need to impose a few well-formedness constraints on systems. We write $\mathsf{fc}(P)$ to denote the set of free constants in P, $\mathsf{fx}(P)$ to denote the set of free variables ranging over expressions, and $\mathsf{fy}(P)$ to denote the set of free variables ranging over terms. For a system of the form displayed above we require that $\mathsf{fx}(P_i) \subseteq \{x_i\}$, $\mathsf{fy}(P_i) = \emptyset$, $\mathsf{fx}(P_*) = \emptyset$, $\mathsf{fy}(P_*) = \emptyset$ (processes are closed), and put no restrictions on $\mathsf{fc}(P_i)$ and $\mathsf{fc}(P_*)$. Finally, for the sake of simplifying the technical developments, we assume that a variable is defined exactly once in the system – a renaming step can be enforced on systems not fulfilling this constraint before executing the analysis.

4.2 Reduction Semantics

The semantics consists of a structural congruence and a transition relation. The *structural congruence* $P_1 \equiv P_2$ is defined in Table 4.2 and expresses when two processes, P_1 and P_2, are congruent to each other. It enforces that processes constitute a commutative monoid with respect to parallel composition and the empty process and it takes care of scope laws for names. Finally, it allows replacement in contexts C given by:

$$C ::= [\,] \;\mid\; (\nu c)\,C \;\mid\; C \,|\, P \;\mid\; P \,|\, C$$

As usual, we apply α-conversion whenever needed in order to avoid accidental capture of names during substitution, and we assume that processes are *equal* up to α-renaming.

For the sake of simplifying the technical proofs, we deviate from the original presentation of the calculus and move the unfolding of recursive calls from the

Table 4.3: The evaluation $\triangleright$ of terms into data and expressions into optional data.

$$c \triangleright c \qquad \frac{t_1 \triangleright c_1 \quad \cdots \quad t_n \triangleright c_n}{g(t_1, \ldots, t_n) \triangleright c} \text{ if } [\![g]\!](c_1, \ldots, c_n) = c$$

$$\mathsf{none} \triangleright \mathsf{none} \qquad \frac{t \triangleright c}{\mathsf{some}(t) \triangleright \mathsf{some}(c)}$$

$$\frac{e_1 \triangleright o_1 \quad \cdots \quad e_n \triangleright o_n}{f(e_1, \ldots, e_n) \triangleright o} \text{ if } [\![f]\!](e_1, \ldots, e_n) = o$$

structural congruence to the reduction rules (cf. § 4.5).

The *transition relation*

$$P \longrightarrow P'$$

describes when a process P evaluates into another process P'. It is parametrised on the relation $t \triangleright c$ describing when a term t evaluates to a constant c and the relation $e \triangleright o$ describing when an expression e evaluates to a constant optional data o that either has the form $\mathsf{some}(c)$ or is none; the definitions of these relations are in Table 4.3, where we write $[\![g]\!](c_1, \ldots, c_n)$ for the application of function g to actual parameters, and likewise for functions f. Furthermore, we make use of two auxiliary relations

$$c_1!c_2 \vdash b \to b'$$

for specifying the effect on the binder b of matching the output $c_1!c_2$, and

$$b ::_v \theta$$

for recording (in $v \in \{\mathsf{tt}, \mathsf{ff}\}$) whether or not all required inputs of b have been performed as well as information about the substitution (θ) that has been constructed. To formalise this we extend the syntax of binders to include substitutions

$$b ::= \cdots \mid [\mathsf{some}(c)/x]$$

where $[\mathsf{some}(c)/x]$ is the substitution that maps x to $\mathsf{some}(c)$ and leaves all other variables unchanged. We write id for the identity substitution and $\theta_2 \circ \theta_1$ for the composition of two substitutions (θ_2 *after* θ_1), so that $(\theta_2 \circ \theta_1)(x) = \theta_2(\theta_1(x))$ for all x.

The first part of Table 4.4 defines the transition relation $P \longrightarrow P'$. Clause (In-ff) expresses that a synchronisation replaces a binder b with a new binder b' recording the output just performed; this transition is only possible when $b' ::_{\mathsf{ff}} \theta$ holds, meaning that more inputs are required before proceeding with

Table 4.4: The reduction semantics $\longrightarrow$ of the Quality Calculus.

$$\frac{t_1 \rhd c_1 \quad t_2 \rhd c_2 \quad c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{ff}} \theta}{t_1!t_2.P_1 \mid b.P_2 \longrightarrow P_1 \mid b'.P_2} \text{(In-ff)}$$

$$\frac{t_1 \rhd c_1 \quad t_2 \rhd c_2 \quad c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{tt}} \theta}{t_1!t_2.P_1 \mid b.P_2 \longrightarrow P_1 \mid P_2\theta} \text{(In-tt)}$$

$$\frac{e \rhd \mathsf{some}(c)}{\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2 \longrightarrow P_1[c/y]} \text{(Case-tt)}$$

$$\frac{e \rhd \mathsf{none}}{\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2 \longrightarrow P_2} \text{(Case-ff)}$$

$$A(e) \longrightarrow P[e/x] \quad \text{if } A(x) \triangleq P \qquad \text{(Rec)}$$

$$\frac{P_1 \equiv P_2 \quad P_2 \longrightarrow P_3 \quad P_3 \equiv P_4}{P_1 \longrightarrow P_4} \text{(Cng)} \qquad \frac{P_1 \longrightarrow P_2}{C[P_1] \longrightarrow C[P_2]} \text{(Cnt)}$$

$$\frac{t_1 \rhd c_1}{c_1!c_2 \vdash t_1?x_2 \to [\mathsf{some}(c_2)/x_2]}$$

$$\frac{c_1!c_2 \vdash b_i \to b_i'}{c_1!c_2 \vdash \&_q(b_1, \cdots, b_i, \cdots, b_n) \to \&_q(b_1, \cdots, b_i', \cdots, b_n)}$$

$$t?x ::_{\mathsf{ff}} [\mathsf{none}/x] \qquad [\mathsf{some}(c)/x] ::_{\mathsf{tt}} [\mathsf{some}(c)/x]$$

$$\frac{b_1 ::_{r_1} \theta_1 \quad \cdots \quad b_n ::_{r_n} \theta_n}{\&_q(b_1, \cdots, b_n) ::_r \theta_n \circ \cdots \circ \theta_1} \text{ where } r = [\![q]\!](r_1, \cdots, r_n)$$

the continuation P_2. Clause (In-tt) considers instead the case where no further inputs are required; this is expressed by the premise $b' ::_{\mathsf{tt}} \theta$. In this case the binding is performed by applying the substitution θ to the continuation process. The subsequent clauses are straightforward; they define the semantics of the case construct, the unfolding of recursive calls, how the structural congruence is embedded in the transition relation, and how transitions take place in contexts.

It is worthwhile observing that the synchronising prefixes in rules (In-) are explicitly mentioned, and this allows to take care of transitions under restrictions in rule (Cnt). Another interpretation of this rule puts the focus of the calculus on closed systems as opposed to open systems, where in the latter possible interactions with the environment are taken care of by a labelled semantics.

The second group of clauses in Table 4.4 defines the auxiliary relation $c_1!c_2 \vdash$

Table 4.5: Quality predicates q and their semantics $[\![q]\!]$.

$$
\begin{array}{rl}
[\![\forall]\!](r_1,\cdots,r_n) = (|\{i \mid r_i = \mathsf{tt}\}| = n) & = r_1 \wedge \cdots \wedge r_n \\
[\![\exists]\!](r_1,\cdots,r_n) = (|\{i \mid r_i = \mathsf{tt}\}| \geq 1) & = r_1 \vee \cdots \vee r_n \\
[\![\exists!]\!](r_1,\cdots,r_n) = (|\{i \mid r_i = \mathsf{tt}\}| = 1) & \\
[\![m/n]\!](r_1,\cdots,r_n) = (|\{i \mid r_i = \mathsf{tt}\}| \geq m) &
\end{array}
$$

$b \to b'$. We have one clause for each of the two syntactic forms of b and the idea is simply to record the binding of the value received in the appropriate position.

The auxiliary relation $b ::_v \theta$ is defined in the final group of clauses in Table 4.4. Here we perform a pass over the (extended) syntax of the binder b, evaluating whether or not a sufficient number of inputs has been performed (recorded in v) and computing the associated substitution θ. Table 4.5 gives examples of quality predicates q to be used in the sequel together with their semantics; here we write $|X|$ for the cardinality of the set X.

Flexible vs. rigid semantics. The semantics of Table 4.4 is a *rigid* semantics: The first time the top-level quality predicate holds the remaining inputs are no longer of interest and the computation can proceed. An alternative would be to use a *flexible* semantics and replace rules (In-ff) and (In-tt) of Table 4.4 with

$$
\frac{t_1 \rhd c_1 \quad t_2 \rhd c_2 \quad c_1!c_2 \vdash b \to b'}{t_1!t_2.P_1 \mid b.P_2 \longrightarrow P_1 \mid b'.P_2} \qquad \frac{b ::_{\mathsf{tt}} \theta}{b.P \longrightarrow P\theta}
$$

The first clause expresses that we may continue accepting inputs even when $b ::_{\mathsf{tt}} \theta$ holds, that is, after the top-level quality condition is met the first time. The second clause ensures that at any point where the quality condition is met we can decide to proceed with the continuation process. Hence, there is a non-deterministic choice as to how many inputs are accepted beyond the minimum number. This becomes a bit tricky when using quality predicates that do not satisfy a monotonicity requirement, meaning that the quality condition may go from true to false once more inputs have been accepted; this is for example the case for $\exists!$ in Table 4.5. On top of this important difference between the rigid and the flexible semantics, they also differ in their "speed"; as an example, in the rigid semantics a single step is needed to perform the binding of a single input, whereas two steps are needed in the flexible semantics. Clearly, the *flexible* semantics admits all the behaviours of the *rigid* semantics as well as additional ones.

4.3 Expressiveness

The binding operator $\&_q(b_1,\cdots,b_n)$ is surprisingly powerful. In the following, we shall show how the primitives of the Quality Calculus can be used to define

a number of other constructs known from process calculi. In the other direction the Quality Calculus can be encoded into the π-calculus, but it would seem that some binding operators would require an exponential expansion; as an example, $\&_{n/2n}(b_1, \cdots, b_{2n})$, indicating that half of the $2n$ arguments are needed, would seem to require that the π-calculus encoding would need to enumerate subsets of $\{1, \cdots, 2n\}$ with at most n elements.

Guarded sum. Let us consider the guarded sum $\Sigma_{i=1}^{n} t_i?x_i.P_i$ of input-guarded processes. It can easily be encoded in our calculus using the binding construct:

$$\begin{array}{rcl}\Sigma_{i=1}^{n} t_i?x_i.P_i & \triangleq & \&_{\exists}(t_1?x_1, \cdots, t_n?x_n).\\ & & (\mathsf{case}\ x_1\ \mathsf{of}\ \mathsf{some}(y_1)\colon P_1\ \mathsf{else}\ 0 \mid \\ & & \qquad \vdots \\ & & \mid \mathsf{case}\ x_n\ \mathsf{of}\ \mathsf{some}(y_n)\colon P_n\ \mathsf{else}\ 0)\end{array}$$

Here the quality predicate $\exists$ expresses that only 1 of the n inputs is required and we assume that no x_i occurs free in P_j when $i \neq j$.

Generalised input binder. We now introduce a version of the binding operator that always honour all its sub-inputs, even though it does not need all of them, thereby ensuring that other processes will not become stuck for they cannot synchronise. The new binding operator is written $\&_q^?(t_1?x_1, \cdots, t_n?x_n)$ and is defined by

$$\begin{array}{rcl}\&_q^?(t_1?x_1, \cdots, t_n?x_n).P & \triangleq & \&_q(t_1?x_1, \cdots, t_n?x_n).\\ & & (P \mid \mathsf{case}\ x_1\ \mathsf{of}\ \mathsf{some}(y_1)\colon 0\ \mathsf{else}\ t_1?x_1 \\ & & \qquad \vdots \\ & & \mid \mathsf{case}\ x_n\ \mathsf{of}\ \mathsf{some}(y_n)\colon 0\ \mathsf{else}\ t_n?x_n)\end{array}$$

The idea is to spawn processes in parallel to the continuation P taking care of the inputs that were not necessary according to the quality predicate.

Internal non-deterministic choice. We now show how to encode a version of the general sum $\bigoplus_{i=1}^{n} P_i$ of processes modelling *internal non-deterministic choice* between the alternatives. The idea is to introduce n fresh channels d_i over which a fresh constant d is communicated and bound to fresh variables x_i and y_i, and then to select one of the summands:

$$\begin{array}{rcl}\bigoplus_{i=1}^{n} P_i & \triangleq & (\nu d_1) \cdots (\nu d_n)\,(\nu d)\\ & & (d_1!d \mid \cdots \mid d_n!d\\ & & \mid \&_{\exists}(d_1?x_1, \cdots, d_n?x_n).\\ & & \qquad (\mathsf{case}\ x_1\ \mathsf{of}\ \mathsf{some}(y_1)\colon P_1\ \mathsf{else}\ d_1?x_1 \mid \\ & & \qquad\quad \vdots \\ & & \qquad \mid \mathsf{case}\ x_n\ \mathsf{of}\ \mathsf{some}(y_n)\colon P_n\ \mathsf{else}\ d_n?x_n))\end{array}$$

The difference from the ordinary CCS sum is that the choices are not made according to the availability of inputs; rather, an internal non-deterministic choice is performed as in CSP. We use the symbol d instead of c to stress that such channels are introduced by the encoding.

Generalised output prefix. Finally, we introduce an operator that allows a process to learn which outputs have been delivered and then use a quality predicate to determine when to proceed. The idea is to introduce new channels that can be used for internal communication when the outputs have been accepted. The new operator is denoted by $\&_q^!(t_1!t'_1, \cdots, t_n!t'_n)$ and it is defined using the binding operator $\&_q^?(\cdots)$ introduced above:

$$\begin{aligned}\&_q^!(t_1!t'_1, \cdots, t_n!t'_n).P \quad \triangleq \quad & (\nu d_1)\cdots(\nu d_n)\,(\nu d)\\ & (t_1!t'_1.d_1!d \mid \cdots \mid t_n!t'_n.d_n!d\\ & \mid \&_q^?(d_1?x_1, \cdots, d_n?x_n).P)\end{aligned}$$

Here we assume that the new constants and variables do not occur in the terms t_i and t'_i nor in the process P. This operator will ensure that the continuation process P can start when some of the outputs have taken place (as determined by the quality predicate q) and it will also ensure that remaining outputs are still ready to be performed so that other processes do not get stuck because of missing communication possibilities.

We could term this behaviour "stateful transfer", as opposed to the concept of oblivious transfer formulated in computer security.

4.4 A Robust Base Station

We consider a fragment of a Wireless Sensor Network application inspired by [AIL05], where a base station BS communicates with a sensor node SN to obtain the value of a physical parameter, which has to be forwarded to a central aggregating unit CU. In order to ease the presentation, we shall take the liberty to use a polyadic version of the calculus and we shall rely on the derived operators just defined in § 4.3.

The sensor node SN is defined by

$$\begin{aligned}\mathsf{SN} \quad \triangleq \quad & \mathsf{0} \oplus (\mathsf{sn}?(x_i, x_m).\\ & \mathsf{case}\ x_i\ \mathsf{of}\ \mathsf{some}(y_i)\mathsf{:}\\ & \qquad \mathsf{case}\ x_m\ \mathsf{of}\ \mathsf{some}(y_m)\mathsf{:}\ y_i!\mathsf{value}(y_m).\mathsf{SN}\ \mathsf{else}\ \mathsf{0}\\ & \mathsf{else}\ \mathsf{0})\end{aligned}$$

A basic node is equipped with a sensor able to measure one or more physical parameters (e.g., temperature, radioactivity) and a transceiver. As a node is typically powered by batteries, at some point in time it will die: this behaviour is captured by the possibility of evolving to 0 non-deterministically in the first line.

While the node is alive, it waits for a request from the base station on channel sn, expecting the identity x_i of the sender and the name x_m of the parameter to be measured. The subsequent case constructs are necessary to extract the actual data, and then the measure is taken and communicated to the base station; the two else branches are in fact not reachable. The function value (which takes data as input and returns data) produces the result of measuring the intended parameter.

The base station will ask the sensor node to measure a physical parameter, and in the interest of its robustness we extend it with a process representing a local computer, able to estimate such a value. The local estimate will be communicated to the central unit and used whenever the sensor node does not respond. The local computer is defined by

$$\mathsf{LC} \triangleq \mathsf{lc}?x_e.\mathsf{case}\ x_e\ \mathsf{of}\ \mathsf{some}(y_e) : \mathsf{lc}!\mathsf{guess}(y_e).\mathsf{LC}\ \mathsf{else}\ 0$$

and it uses the function guess (taking data and returning data) to estimate the value of the intended parameter; again, the case construct is used to extract the actual request and the else branch is not reachable.

The base station will put a limit on how long it will wait for a measure. In order to model this behaviour we make use of a time counter defined by

$$\mathsf{Clock} \triangleq \mathsf{set}?x_t.\mathsf{tick}!\checkmark.\mathsf{Clock}$$

where channel set is used to set a time-out, and the output of the constant $\checkmark$ signals that the prescribed amount of time has passed.

Finally, the base station is defined by the process

$$\begin{aligned}\mathsf{BS} \triangleq\ & (\nu id)\,(\nu m)\,(\nu t)\,\&^{!}_{\exists}(\mathsf{lc}!m, \mathsf{sn}!(id, m)).\mathsf{set}!t \\ & \&_{\forall}(\mathsf{tick}?x_t, \&^{?}_{\exists}(\mathsf{lc}?x_l, id?x_r)) \\ & \mathsf{case}\ x_r\ \mathsf{of}\ \mathsf{some}(y_r) : {}^{1}\mathsf{cu}!y_r.\mathsf{BS}\ \mathsf{else} \\ & \quad \mathsf{case}\ x_l\ \mathsf{of}\ \mathsf{some}(y_l) : {}^{2}\mathsf{cu}!y_l.\mathsf{BS}\ \mathsf{else}\ {}^{3}0\end{aligned}$$

where we have added labels 1,2,3 for later reference. In the first line, the base station issues a request for measuring a parameter m to the local computer and to the sensor node, identifying itself as id. The timer is set to the constant t as soon as one of the recipients has received the request. The second line waits for the deadline and for at least one value among the local estimate and the real measure. This behaviour is determined by the top-most quality predicate $\forall$, which requires that both inputs are successful, and by the inner quality predicate $\exists$, which insists that at least one of its two sub-inputs be successful. As we are using the binding operator $\&^{?}_{\exists}(\dots)$, the other input will be handled when (and if) it arrives. It is important to note that it is also possible that both values arrive before the time has passed. The third line tests whether or not the sensor node responded; if this is the case (label 1) the value is communicated to the central unit, otherwise (label 2) the local estimate is sent. Observe that

in this formalisation the final else branch (labelled 3) is not reachable, for the requests built by the base station correctly match the inputs of SN and LC, and the latter is assumed to respond always, the component being local.

The main process of the system is defined as

$$P_* = (\nu \mathsf{sn})\,(\nu \mathsf{lc})\,(\nu \mathsf{set})\,(\nu \mathsf{tick})\,(\nu \checkmark)\,(\mathsf{BS} \mid \mathsf{SN} \mid \mathsf{LC} \mid \mathsf{Clock})$$

Discussion. Let us conclude by discussing two alternative choices for the binding construct in the second line of BS. One possibility is to use the binder

$$\&_{2/3}(\mathsf{tick}?x_t, \mathsf{lc}?x_l, id?x_r)$$

and this would require that at least one entity among the sensor network and the local computer has communicated a value before proceeding. Observe that we may proceed before the deadline has expired. Another possibility is to use

$$\&_{\exists}(\mathtt{tick}?x_t, \&^{?}_{\exists}(\mathtt{lc}?x_l, id?x_r))$$

and in this case we might end up having no value at all.

4.5 An Explicit Substitution Semantics

So far we have traced the original presentation of the Quality Calculus, resorting to a standard reduction semantics (§ 4.2) where substitutions are applied *directly*, in a form that is customary for process algebras. A reduction semantics clearly retains a superior elegance for presentation purposes, but induces a rather byzantine formulation of the correctness statement of the robustness analysis, which is fundamentally informed by the notion of substitution. A semantics with *explicit substitutions* is required to formally capture the intention of the analysis, which rests on a precise relation between logical formulae describing program points (analysis) and substitutions that arise at those points (semantics).

In the following, we present a semantics of the Quality Calculus with explicit substitutions, and in A.2 we shall show its equivalence to the reduction semantics of § 4.2. The usefulness of an explicit substitution semantics is two-fold: on the one hand, it simplifies the correctness proof of the robustness analysis, since substitutions are not directly applied to processes, thus preserving the syntactical identity of a program point through an execution; on the other hand, the new semantics mimics the execution of the calculus on an abstract machine, and thus it is closer to an implementation. Observe that the new semantics with explicit substitutions is of the reduction kind too, being not labelled, but for the sake of clarity we shall refer to it as "the explicit substitution semantics", retaining the phrase "reduction semantics" for the one of § 4.2.

The overall structure in terms of semantic relations is inherited from the reduction semantics, whereas some judgements are updated in order to keep

substitutions distinct from processes. First of all, the syntax of processes is updated with a new syntactic category S for *explicit processes*, denoting processes active under given substitutions:

$$S \ ::= \ \{\rho\}P \mid (\nu c)\, S \mid b.S \mid t_1!t_2.S \mid S_1|S_2 \mid \mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon S_1\ \mathsf{else}\ S_2$$

where in $\{\rho\}P$ process P is active under substitution ρ, and we write $\mathsf{fc}(\{\rho\}P)$ for the names that are free in P or in ρ, that is, $\mathsf{fc}(\{\rho\}P) = \mathsf{fc}(P) \cup \mathsf{dom}(\rho) \cup \mathsf{rng}(\rho)$. The definition of system is thus updated as

$$\begin{array}{ll} \mathsf{define} & A_1(x_1) \triangleq P_1 \\ & \vdots \\ & A_n(x_n) \triangleq P_n \\ \mathsf{in} & \{\mathsf{id}\}P_* \end{array}$$

(recall that id denotes the identical substitution). In the following, we shall assume that substitutions are total and behave like the identity if not otherwise specified.

A substitution ρ maps variables $x \in \mathcal{X}$ to constant optional data $o \in \mathcal{O}$, variables $y \in \mathcal{Y}$ and names to names $c \in \mathcal{C}$. Given a substitution

$$\rho = [o_1/x_1, \ldots, o_m/x_m, c'_1/y_1, \ldots, c'_n/y_n, c''_1/c_1, \ldots, c''_p/c_p]$$

we call *range* of ρ the set $\mathsf{rng}(\rho) = \{o_1, \ldots, o_m, \ldots, c'_1, \ldots, c'_n, c''_1, \ldots, c''_p\}$ and *domain* of ρ the set $\mathsf{dom}(\rho) = \{x_1, \ldots, x_m, \ldots, y_1, \ldots, y_n, c_1, \ldots, c_p\}$ of variables and names which are not identically mapped by ρ.

As for terms, (ρt) denotes the application of a substitution ρ to a term t. Similarly we write (ρe) for expressions, and (ρb) for binders, where the substitution is applied to the terms contained in b. For the sake of simplifying the technical developments, in the following we assume that the application of a substitution to a term t or expression e takes care of evaluating every function g or f occurring in t or e, that is, (ρt) and (ρe) embed the deterministic evaluations $t \rhd c$ and $e \rhd o$, displayed in Table 4.3.

4.5.1 Directed structural rules

The structural rules for the explicit substitution semantics are displayed in Table 4.6, and consist of *directed* equations on explicit processes. We write $S \Rrightarrow S'$, to stress that the usual symmetry of structural congruences is not part of our axiomatisation, meaning that S is structurally reducible to S', but not necessarily S' is reducible to S. The relation makes use of contexts C on explicit processes, defined as follows:

$$C ::= [\,] \ \mid \ (\nu c)\, C \ \mid \ C|S \ \mid \ S|C$$

Again, we apply α-conversion whenever needed in order to avoid accidental capture of names during substitution, and we assume that explicit processes are *equal* up to α-renaming.

The rules in Table 4.6 can be grouped in two categories. The first group includes standard rules for reflexivity of the congruence (Ref), for defining processes as a commutative monoid with respect to parallel composition (Nil, Nil', Com, Ass), for reordering restrictions (New1), for redundant restrictions (New2, New2'), for scope reduction and extension (New3, New3'), for the transitivity of the congruence (Tra), and for the preservation of the congruence in contexts (Cnt). Observe that this set of rules forms an equivalence relation: all the rules are either self-symmetric, or symmetric versions are explicitly provided or can be derived.

The second group of rules handles substitutions, pushing them to sub-processes ready to execute (S-par, S-new, S-case) as well as mimicking the look-up of bindings in an environment (S-bin, S-out). This is slightly different from other approaches where the look-up procedure is taken care of by the semantics explicitly, but yields a cleaner proof strategy for the correctness of the robustness analysis. Observe that this set of rules does not form an equivalence relation, and in particular we do not want symmetric versions of such rules, that would correspond to "de-istantiate" variables, executing a program in reverse. This is one potential reason why explicit substitution semantics are often presented without a congruence. Nonetheless, observe that omitting symmetry compels taking care of scope extension in some other way. Ferrari et al. [FMQ94] simply assume that no communication is performed on names that are not globally known. Other authors resort to labelled semantics that allow a bound output to communicate with an input and carry the restriction of the communicated name to the continuation processes: this is the case of the rule CLOSE of the π-calculus [Par01], followed for instance in [BJPV11]. A set of directed structural rules facilitates to overcome the assumption of global names while retaining a superior elegance over a transitional scope extrusion, as the semantics is factored in a compact set of reduction rules, governing the executions of processes, and an intuitive set of rewriting rules ruling their structure. We ought to acknowledge, however, that preferring such an approach over labelled transitions is largely a matter of personal taste.

Even if it is improper to call $\Rrightarrow$ a structural congruence, nonetheless we retain this phrase to stress that $\Rrightarrow$ and the standard $\equiv$ fulfil similar roles.

In the wake of [EG04], we say that structural rules are in charge of expressing *static* features of processes, while the semantics accommodates their *dynamic* or *behavioural* features. This being said, a general consensus sustains the opinion that some features can be regarded as either static or dynamic [Par01, EG04], the choice being mainly a matter of taste, mathematical elegance, and convenience in proofs. This is chiefly the case of replication or unfolding of recursive calls, regarded as static by some authors and as dynamic by others. We delegate recursion to the transition system, thus embracing the dynamic point of view.

Table 4.6: The directed structural rules $\Rrightarrow$ of the Quality Calculus.

$$S \Rrightarrow S \quad \text{(Ref)} \qquad S|\{\rho\}\mathsf{0} \Rrightarrow S \quad \text{(Nil)}$$

$$S \Rrightarrow S|\{\rho\}\mathsf{0} \quad \text{(Nil')} \qquad S_1|S_2 \Rrightarrow S_2|S_1 \quad \text{(Com)}$$

$$S_1|(S_2|S_3) \Rrightarrow (S_1|S_2)|S_3 \quad \text{(Ass)}$$

$$(\nu c_1)\,(\nu c_2)\,S \Rrightarrow (\nu c_2)\,(\nu c_1)\,S \quad \text{(New1)}$$

$$(\nu c)\,\{\rho\}P \Rrightarrow \{\rho\}P \text{ if } c \notin \mathsf{fc}(\{\rho\}P) \quad \text{(New2)}$$

$$\{\rho\}P \Rrightarrow (\nu c)\,\{\rho\}P \text{ if } c \notin \mathsf{fc}(\{\rho\}P) \quad \text{(New2')}$$

$$(\nu c)\,(\{\rho_1\}P_1|\{\rho_2\}P_2) \Rrightarrow ((\nu c)\,\{\rho_1\}P_1)|\{\rho_2\}P_2 \text{if } c \notin \mathsf{fc}(\{\rho_2\}P_2) \quad \text{(New3)}$$

$$((\nu c)\,\{\rho_1\}P_1)|\{\rho_2\}P_2 \Rrightarrow (\nu c)\,(\{\rho_1\}P_1|\{\rho_2\}P_2) \text{if } c \notin \mathsf{fc}(\{\rho_2\}P_2) \quad \text{(New3')}$$

$$\frac{S_1 \Rrightarrow S_2 \quad S_2 \Rrightarrow S_3}{S_1 \Rrightarrow S_3}\ \text{(Tra)} \qquad \frac{S_1 \Rrightarrow S_2}{C[S_1] \Rrightarrow C[S_2]}\ \text{(Cnt)}$$

$$\{\rho\}(P_1|P_2) \Rrightarrow \{\rho\}P_1|\{\rho\}P_2 \quad \text{(S-par)}$$

$$\{\rho\}((\nu c)\,P) \Rrightarrow \begin{cases} (\nu c)\{\rho\}P & \text{if } c \notin \mathsf{fc}(\{\rho\}P) \\ (\nu c')\{\rho \circ [c'/c]\}P & \text{if } c \in \mathsf{fc}(\{\rho\}P) \wedge c' \notin \mathsf{fc}(\{\rho\}P) \end{cases} \quad \text{(S-new)}$$

$$\{\rho\}\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2 \Rrightarrow \mathsf{case}\ (\rho e)\ \mathsf{of}\ \mathsf{some}(y)\colon \{\rho\}P_1\ \mathsf{else}\ \{\rho\}P_2 \quad \text{(S-case)}$$

$$\{\rho\}(b.P) \Rrightarrow (\rho b).\{\rho\}P \ \text{(S-bin)} \qquad \{\rho\}(t_1!t_2.P) \Rrightarrow (\rho t_1)!(\rho t_2).\{\rho\}P \ \text{(S-out)}$$

This choice facilitates the formal proofs of A.2, as a structural rule for unfolding of recursion would share traits with both groups in Table 4.6, and giving more than one structural rule to deal with recursion seems not natural.

On directed rules. It is worthwhile observing that the notion of structural congruence has its roots in the *directed* heating and cooling rules of the Chemical Abstract Machine of Berry and Boudol [BB90]. For instance, $S|\mathsf{0} \Rrightarrow S$ models the evaporation of $\mathsf{0}$ upon heating the component, and its not revertible. Inspired by these rules, Milner presented a π-calculus whose semantics is parametrised on structural rules [Mil90] that are *symmetric*, hence yielding an equivalence relation.

Recently, a few works recovered the original directed fashion of structural rules (e.g., [PNR08, RNKP11]), omitting a general law for symmetry and introducing specific clauses for the cases where symmetry is required (e.g., scope laws), while the mainstream approach follows Milner's presentation. We say that the rules of the former approach are directed (or oriented) exactly because of the absence of a general rule for symmetry.

A full historical account of the evolution of the usage of structural congruences in process calculi is far beyond the scope of this dissertation. We limit here to observe that directed rules may be convenient for a number of reasons. First, directing some rules matches more closely our intuition of how the computation evolves: for instance, $\mathsf{0}$ can be thought of the result of a terminated computation, and allowing the introduction of a terminated parallel component by reading (Nil) from right to left (by virtue of symmetry) seems to contradict this intuition. This implementation viewpoint is thoroughly investigated in [GLP04], which introduced the idea of *operational effectiveness criteria* for structural rules. Secondly, as we have mentioned above, considering directed rules allows retaining some kind of structural relation among processes even in presence of explicit substitutions, thus avoiding strong assumptions on bound names or moving to labelled semantics.

This being said, we could not devise an operational effective version of the structural rules, as one of our task was to establish the equivalence of the new semantics and the original reduction semantics, which is parametrised on the standard congruence of the π calculus. It is worthwhile noticing, however, that given an operational effective version of the structural rules it would still be possible to prove the equivalence of the two notions of congruence, up to preliminary and final rewriting of processes by means of the standard rules of $\equiv$. However, this would significantly complicate the technical developments, therefore we limit to point the reader to [SW01, Ch. 1] for a similar proof strategy.

4.5.2 The transition relation

The transition relation with explicit substitutions, presented in Table 4.7, describes how an explicit process $S = \{\rho\}P$ evolves into another explicit process $S' = \{\rho'\}P'$. Like in the reduction semantics case, we rely in the relation $c_1!c_2 \vdash b \to b'$ for specifying the effect on the binder b of matching the output $c_1!c_2$, and on the relation $b ::_r \theta$ for evaluating binders and recording substitutions. Observe that in general substitutions ρ differ from substitutions θ, in that the latter only map variables $x \in \mathcal{X}$ to optional data.

The rules in Table 4.7 define the transition relation $S \longrightarrow S'$ retracing closely the reduction semantics. Observe that terms and expressions are now evaluated when pushing substitutions inside a process by means of structural rules, hence the premises of base cases are simplified. Moreover, note that in rule (Rec) x is the only variable free in P, and thus the substitution on the right-hand side is simplified so as to consider only x. Finally, rule (Cng) embeds the new

Table 4.7: The transition relation $\longrightarrow$ of the Quality Calculus with explicit substitutions.

$$\frac{c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{ff}} \theta}{c_1!c_2.\{\rho_1\}P_1 \mid b.\{\rho_2\}P_2 \longrightarrow \{\rho_1\}P_1 \mid b'.\{\rho_2\}P_2} \quad \text{(In-ff)}$$

$$\frac{c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{tt}} \theta}{c_1!c_2.\{\rho_1\}P_1 \mid b.\{\rho_2\}P_2 \longrightarrow \{\rho_1\}P_1 \mid \{\theta \circ \rho_2\}P_2} \quad \text{(In-tt)}$$

$$\mathsf{case\ some}(c)\ \mathsf{of\ some}(y)\colon \{\rho_1\}P_1\ \mathsf{else}\ \{\rho_1\}P_2 \longrightarrow \{[c/y] \circ \rho_1\}P_1 \quad \text{(Case-tt)}$$

$$\mathsf{case\ none\ of\ some}(y)\colon \{\rho_2\}P_1\ \mathsf{else}\ \{\rho_2\}P_2 \longrightarrow \{\rho_2\}P_2 \quad \text{(Case-ff)}$$

$$\{\rho\}A(e) \longrightarrow \{[(\rho e)/x]\}P \quad \text{if } A(x) \triangleq P \quad \text{(Rec)}$$

$$\frac{S_1 \Rrightarrow S_2 \quad S_2 \longrightarrow S_3 \quad S_3 \Rrightarrow S_4}{S_1 \longrightarrow S_4} \quad \text{(Cng)} \qquad \frac{S \longrightarrow S'}{C[S] \longrightarrow C[S']} \quad \text{(Cnt)}$$

structural congruence in the transition relation.

The relation between output and binders and the evaluation of binders is left unchanged and inherited from Table 4.4.

Flexible vs. rigid semantics. Like in the case of the reduction semantics, a more *flexible* explicit substitution semantics is obtained by replacing rules (In-ff) and (In-tt) of Table 4.7 with

$$\frac{c_1!c_2 \vdash b \to b'}{c_1!c_2.\{\rho_1\}P_1 \mid b.\{\rho_2\}P_2 \longrightarrow \{\rho_1\}P_1 \mid b'.\{\rho_2\}P_2} \qquad \frac{b ::_{\mathsf{tt}} \theta}{b.\{\rho\}P \longrightarrow \{\theta \circ \rho\}P}$$

The robustness analysis of § 4.6.1 is expressive enough to capture the richer flexible semantics, but the correctness proof concerns the more robust rigid semantics. Hence, in the following we shall limit our discussion to the rigid semantics.

4.6 Robustness: Absence of Communication

4.6.1 Robustness analysis

The Quality Calculus provides the means for expressing due care in always having default data available in case the real data cannot be obtained, but – recall Dijkstra's quote in § 2.2 – it does not enforce it.

Our enforcement mechanism will be a SAT-based robustness analysis for characterising whether or not variables over optional data do indeed contain data. The analysis attaches propositional formulae to all points of interest in the processes; the formulae characterise the combinations of optional data that could be missing. At key places one would like to demand that such formulae would always require default data to be available; this translates into demanding that certain logical formulae are unsatisfiable as determined by a SAT-solver.

The formulae generated by the analysis encode optional data as Boolean values in the following manner. A value of the form $\mathsf{some}(\cdot)$ is coded as tt and none is coded as ff. In the following, we shall write $\overline{o}$ to denote the Boolean encoding of the optional data o, i.e., $\overline{\mathsf{some}(\cdot)} = \mathsf{tt}$ and $\overline{\mathsf{none}} = \mathsf{ff}$. As an example, the formula $x_1 \vee (x_2 \wedge x_3)$ indicates that either x_1 is available or both x_2 and x_3 are available, the variables ranging over Boolean values.

The judgements. The main judgement of our analysis takes the form

$$\vdash \varphi \mathbin{@} P$$

and the idea is that the formula φ describes the program point immediately before P. This is ambiguous in case there are multiple occurrences of the same sub-process in the system: the traditional solution is to add labels to disambiguate such occurrences, but we dispense with this in order not to complicate the notation. The intended semantic interpretation of this judgement is that

$$\text{if } \vdash \varphi \mathbin{@} P \text{ and } \{\mathsf{id}\}P_* \longrightarrow^* \{\rho\}P \text{ then } \overline{\rho} \models \varphi$$

where $\overline{\rho}$ is the mapping obtained by point-wise application of the encoding $\overline{\cdot}$, $\overline{\rho} \models \varphi$ denotes the truth of φ under the interpretation $\overline{\rho}$ (the notation is defined formally in A.1), and $\longrightarrow^*$ denotes the reflexive and transitive closure of $\longrightarrow$. In a nutshell, the correctness result states that reachability of program points entails satisfiability of the corresponding formulae, and by contra-position that *unsatisfiability entails unreachability.*

We shall make use of two auxiliary judgements. One is for bindings

$$\vdash b \blacktriangleright \varphi$$

and the idea is that the formula φ describes the bindings of the variables that correspond to passing the binder b successfully. The intended semantic interpretation of this judgement is that

$$\text{if } \vdash b \blacktriangleright \varphi \text{ and } b ::_{\mathsf{tt}} \theta \text{ then } \overline{\theta} \models \varphi$$

(cf. Lemma A.1.2 in A.1). The other auxiliary judgement is for expressions; it takes the form

$$\vdash e \triangleright \varphi$$

and the idea is that the formula φ describes the result of evaluating the expression e. The intended semantic interpretation of this judgement is that

Table 4.8: The robustness analysis $\vdash \varphi \mathbin{@} P$ of the Quality Calculus.

$$\vdash \mathsf{tt} \mathbin{@} P_* \qquad \vdash \mathsf{tt} \mathbin{@} P_1 \qquad \cdots \qquad \vdash \mathsf{tt} \mathbin{@} P_n$$

$$\frac{\vdash \varphi \mathbin{@} (\nu c)\, P}{\vdash \varphi \mathbin{@} P} \qquad \frac{\vdash \varphi \mathbin{@} (P_1 \mid P_2)}{\vdash \varphi \mathbin{@} P_1} \qquad \frac{\vdash \varphi \mathbin{@} (P_1 \mid P_2)}{\vdash \varphi \mathbin{@} P_2}$$

$$\frac{\vdash \varphi \mathbin{@} (b.P) \quad \vdash b \blacktriangleright \varphi_b}{\vdash \varphi \wedge \varphi_b \mathbin{@} P} \qquad \frac{\vdash \varphi \mathbin{@} (t_1!t_2.P)}{\vdash \varphi \mathbin{@} P}$$

$$\frac{\vdash \varphi \mathbin{@} (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y) \colon P_1\ \mathsf{else}\ P_2) \quad \vdash e \triangleright \varphi_e}{\vdash \varphi \wedge \varphi_e \mathbin{@} P_1}$$

$$\frac{\vdash \varphi \mathbin{@} (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y) \colon P_1\ \mathsf{else}\ P_2) \quad \vdash e \triangleright \varphi_e}{\vdash \varphi \wedge \neg\varphi_e \mathbin{@} P_2}$$

$$\frac{\vdash \varphi \mathbin{@} P}{\vdash \varphi' \mathbin{@} P} \quad \text{if } \varphi \Leftrightarrow \varphi'$$

$$\vdash t?x \blacktriangleright x \qquad \frac{\vdash b_1 \blacktriangleright \varphi_1 \quad \cdots \quad \vdash b_n \blacktriangleright \varphi_n}{\vdash \&_q(b_1, \cdots, b_n) \blacktriangleright [\![q]\!](\varphi_1, \cdots, \varphi_n)}$$

$$\vdash x \triangleright x \qquad \vdash \mathsf{some}(t) \triangleright \mathsf{tt} \qquad \vdash \mathsf{none} \triangleright \mathsf{ff}$$

$$\frac{\vdash e_1 \triangleright \varphi_1 \quad \cdots \quad \vdash e_n \triangleright \varphi_n}{\vdash f(e_1, \cdots, e_n) \triangleright [\![f]\!](\varphi_1, \cdots, \varphi_n)}$$

$$\text{if } \vdash e \triangleright \varphi \ \text{ and } \ e \rhd o \ \text{ then } \ \overline{o} \models \varphi$$

(cf. Lemma A.1.1 in A.1).

The detailed definition. The formal definition of $\vdash \varphi \mathbin{@} P$ is given by the inference system in the top-most part of Table 4.8. It operates in a top-down manner (as opposed to a more conventional bottom-up manner) and gets started by an axiom $\vdash \mathsf{tt} \mathbin{@} P_*$ for the main process, stating that it is reachable. Moreover, we have an axiom for each of the processes defined in the system under study; they have the form $\vdash \mathsf{tt} \mathbin{@} P_i$, thereby ensuring that the process definitions are analysed in all contexts.

The first inference rule expresses that if φ describes the program point just before a process of the form $(\nu c)\, P$, then it also describes the program point just before P. Then we have two rules for parallel composition: if φ describes the program point before $P_1|P_2$, then it also describes the program point just before each of the two processes. The rule for bindings is more interesting; here

we make use of the auxiliary analysis judgement $\vdash b \blacktriangleright \varphi_b$, explained below, for analysing the binder b. The information φ describing the program point before $b.P$ is transformed into $\varphi \wedge \varphi_b$ in order to describe the program point before P, accounting for the satisfaction condition of the binder b. The rule for output should now be straightforward. The two rules for the case construct make use of the auxiliary analysis judgement $\vdash e \triangleright \varphi_e$, explained below, for analysing expression e; this gives rise to a formula describing the outcome of the test being performed and this information is added to describe the program point just before the selected branch.

In the case of binders the formula φ produced by the judgement $\vdash b \blacktriangleright \varphi$ characterises the availability of data as provided by the binder upon consuming it. In the detailed definition of $\vdash b \blacktriangleright \varphi$, presented in the second part of Table 4.8, we rely on the formula schemes $[\![q]\!](r_1, \cdots, r_n)$ of Table 4.5 for encoding the effect of quality predicates q.

The last part of Table 4.8 defines the judgement $\vdash e \triangleright \varphi$ for expressions and, as already mentioned, the idea is that the formula φ characterises the availability of data used in e. Also here we rely on formula schemes of the form $[\![f]\!](r_1, \cdots, r_n)$ for encoding the effect of functions f and we assume that they satisfy the following soundness and completeness property:

$$[\![f]\!](\overline{o_1}, \cdots, \overline{o_n}) = \overline{o} \quad \text{whenever} \quad f(o_1, \cdots, o_n) \rhd o$$

With respect to the original robustness analysis of [RNV12] we dispense with existential quantifiers in formulae to capture the scope of variables, as justified by the restriction to processes where variables are bound exactly once.

4.6.2 Analysing the base station

Let us return to the base station BS of § 4.4 where we now want to compute the analysis results for the program points identified by the three labels. Starting with $\vdash \mathsf{tt} \,@\, \mathsf{BS}$ we obtain the following formulae at the labels:

$$\begin{array}{ll} 1: & (x_1 \vee x_2) \wedge x_t \wedge (x_l \vee x_r) \wedge x_r \\ 2: & (x_1 \vee x_2) \wedge x_t \wedge (x_l \vee x_r) \wedge (\neg x_r) \wedge x_l \\ 3: & (x_1 \vee x_2) \wedge x_t \wedge (x_l \vee x_r) \wedge (\neg x_r) \wedge (\neg x_l) \end{array}$$

where we used the same variable names used in the process in order to stress the relationship between the formulae produced by the analysis and the program points they describe, even if here the variables range over the Boolean encoding of optional data. Observe that $(x_1 \vee x_2)$ refers to the generalised output prefix $\&^{!}_{\exists}(\mathsf{lc}!p, \mathsf{sn}!(id, p))$ encoded as shown in § 4.3, $(x_t \wedge (x_l \vee x_r))$ is the condition for passing the quality binder in the second line, and the remainder identifies the condition for reaching the given label. We can then ask whether or not the process points decorated with labels may be reachable, that is, whether or not

the corresponding formulae are satisfiable. We obtain the following satisfying substitutions:

$$\begin{array}{ll} 1: & [x_1 \mapsto \mathsf{tt}; x_2 \mapsto \mathsf{ff}; x_l \mapsto \mathsf{tt}; x_r \mapsto \mathsf{tt}; x_t \mapsto \mathsf{tt}] \\ 2: & [x_1 \mapsto \mathsf{tt}; x_2 \mapsto \mathsf{ff}; x_l \mapsto \mathsf{tt}; x_r \mapsto \mathsf{ff}; x_t \mapsto \mathsf{tt}] \\ 3: & \text{unsatisfiable} \end{array}$$

This shows that the occurrence of process $\mathsf{0}$ in BS at label 3 will never be executed.

Let us conclude by considering the variants of the base station discussed at the end of § 4.4. Using the binder $\&^?_{2/3}(\mathsf{tick}?x_t, \mathsf{lc}?x_l, id?x_r)$ we get slightly different formulae but the satisfiability results are the same as above: the formula for the process labelled 3 is unsatisfiable whereas the others have satisfying assignments. Using the binder $\&_\exists(\mathsf{tick}?x_t, \&^?_\exists(\mathsf{lc}?x_l, id?x_r))$ we get the following formula for the process labelled 3:

$$3: \quad (x_1 \vee x_2) \wedge (x_t \vee x_l \vee x_r) \wedge (\neg x_r) \wedge (\neg x_l)$$

which is satisfiable using the substitution:

$$3: \quad [x_2 \mapsto \mathsf{ff}; x_1 \mapsto \mathsf{tt}; x_l \mapsto \mathsf{ff}; x_r \mapsto \mathsf{ff}; x_t \mapsto \mathsf{tt}]$$

The occurrence of process $\mathsf{0}$ labelled 3 might thus be reachable. The above substitution gives us an indication of when this may happen: the binder

$$\&_\exists(\mathsf{tick}?x_t, \&^?_\exists(\mathsf{lc}?x_l, id?x_r))$$

will be successful when $x_t = \mathsf{tt}$ meaning that the time has passed but it does not need to be the case that any of the schedules are available as reflected by $x_l = \mathsf{ff}$ and $x_g = \mathsf{ff}$. In this case the terminated process $\mathsf{0}$ at label 3 will in fact be reached and BS will terminate.

We have implemented this analysis by writing a program in Standard ML for computing the formulae at the program points of interest, and used the SAT and SMT solver Z3 (cf. § 2.3) to determine whether or not the formulae are satisfiable. For the examples we have studied the answer is obtained in less than a second on an ordinary laptop computer.

4.6.3 Formal correctness

The proof of correctness is organised as follows:

1. the robustness analysis of § 4.6.1 is proven sound with respect to the explicit substitution semantics;

2. the explicit substitution semantics is proven equivalent to the reduction semantics of § 4.2.

In this section we present the key statements of the correctness result. Auxiliary lemmata and proofs are deferred to A.1, while A.2 proves the semantic equivalence.

For the sake of easing the presentation, we shall first introduce some notation and then two main results concerning the structural congruence and the transition relation of the semantics, from which the correctness theorem follows immediately.

It is worthwhile observing that the rules of the analysis being deterministic, if two different formulae are derived for a given process, then it must be the case that such formulae are in a bi-implication relation.

Fact 4.6.1 $(\vdash \varphi \mathbin{@} P \wedge \vdash \varphi' \mathbin{@} P) \Rightarrow (\varphi \Leftrightarrow \varphi')$

Therefore, we can define the formula Φ_P that characterises the reachability of process P as follows:

$$\Phi_P = \begin{cases} \varphi & \text{if } \vdash \varphi \mathbin{@} P \\ \mathsf{undef} & \text{if } \nexists \varphi \,.\, \vdash \varphi \mathbin{@} P \end{cases}$$

where it does not matter which φ we choose in the first case, as all formulae derived for P are equivalent, whereas the second case only occurs when P is not a sub-process of the main process P_* of the system under study. It is worthwhile observing that restricting our attention to Φ_P is sound, since the equivalence of φ and φ' in Fact 4.6.1 entails their equisatisfiability, and truth models is all the robustness analysis is concerned with.

Moreover, in the proofs we shall deal with step-wise evaluation of binders.

Definition 4.1 $(\theta \gg b)$ Let θ be a substitution and b a binder. Function $\theta \gg b$ computes the binder obtained by instantiating b according to θ:

$$\theta \gg t?x = \begin{cases} t?x & \text{if } x \notin \mathsf{dom}(\theta) \text{ or } (\theta x) = \mathsf{none} \\ [(\theta x)/x] & \text{otherwise} \end{cases}$$
$$\theta \gg [\mathsf{some}(c)/x] = [\mathsf{some}(c)/x]$$
$$\theta \gg \&_q(b_1, \ldots, b_n) = \&_q(\theta \gg b_1, \ldots, \theta \gg b_n)$$

In particular, notice that $\theta \gg b$ gives rise to a binder, which can be evaluated by means of $::_v$ according to the semantic rules of Table 4.4. Moreover, observe that variables are mapped to none only when a binder is satisfied and the continuation process is thus ready to execute, therefore $\theta \gg t?x$ never maps input variables to none.

As a consequence of evaluating binders step-wise, we need to extend the judgement $\vdash b \blacktriangleright \varphi_b$ of the analysis to the case of partially-evaluated binders:

$$\vdash [\mathsf{some}(c)/x] \blacktriangleright x$$

where this choice conforms with the formula that is produced for $t?x$, that is, the non-instantiated version of the same binder.

As an inspection of Table 4.6 and 4.7 highlights, the interplay between the directed congruence and the explicit substitution semantics yields explicit processes with a very precise shape, formalised as follows.

Definition 4.2 (Goodness) Let S be an explicit process. S is *good*, denoted $\mathsf{good}(S)$, if

$$
\begin{array}{l}
((\exists C.S = C[\{\rho\}P]) \Rightarrow \overline{\rho} \models \Phi_P) \wedge \\
((\exists C.S = C[t_1!t_2.\{\rho\}P]) \Rightarrow \overline{\rho} \models \Phi_P) \wedge \\
((\exists C.S = C[\mathsf{case\ some}(c)\ \mathsf{of\ some}(y)\colon \{\rho\}P\ \mathsf{else}\ \{\rho\}P_2]) \Rightarrow \overline{[c/y] \circ \rho} \models \Phi_P) \wedge \\
((\exists C.S = C[\mathsf{case\ none\ of\ some}(y)\colon \{\rho\}P_1\ \mathsf{else}\ \{\rho\}P]) \Rightarrow \overline{\rho} \models \Phi_P) \wedge \\
(\forall \Theta \forall \theta.((\exists C.S = C[b.\{\rho\}P] \ \wedge\ (\Theta \gg b) ::_{\mathsf{tt}} \theta) \Rightarrow \overline{\theta \circ \rho} \models \Phi_P)).
\end{array}
$$

The main correctness result is stated in Theorem 4.6.1 below, the core idea being that goodness is preserved in a semantic evaluation: if a process P is active under substitution ρ in the semantics and its reachability is described by Φ_P in the analysis, then $\overline{\rho}$ is a model for Φ_P. Equivalently, by contraposition, if a formula is unsatisfiable then there is no derivation in the semantics leading to the corresponding program point.

Theorem 4.6.1 (Correctness of the robustness analysis) *For all systems*

$$
\begin{array}{ll}
\mathsf{define} & A_1(x_1) \triangleq P_1 \\
 & \vdots \\
 & A_n(x_n) \triangleq P_n \\
\mathsf{in} & \{\mathsf{id}\}P_*
\end{array}
$$

it holds that

$$\forall S\,.\,(\{\mathsf{id}\}P_* \longrightarrow^* S \ \Rightarrow\ \mathsf{good}(S))$$

Proof. The proof is organised by induction on the length k of the derivation sequence $\{\mathsf{id}\}P_* \longrightarrow^* S$.

Basis. If $k = 0$, then it is $S = \{\mathsf{id}\}P_*$ and $\Phi_{P_*} = \mathsf{tt}$, from which the result follows since $\mathsf{id} \models \mathsf{tt}$.

Step. Assume that the result holds for $k \leq k_0$; we shall prove it for $k_0 + 1$. The whole derivation sequence can be written as

$$\{\mathsf{id}\}P_* \longrightarrow^{k_0} S' \longrightarrow S$$

The inductive hypothesis applies to the first k_0 steps of the derivation, leading to

$$\mathsf{good}(S')$$

Therefore, we shall now show

$$S' \longrightarrow S \wedge \mathsf{good}(S') \;\Rightarrow\; \mathsf{good}(S)$$

that is, the last derivation step in the sequence preserves the goodness of the system. The result follows directly from Lemma 4.6.2. □

The main technical results on which the theorem rests establish that the structural congruence and the explicit substitution semantics of the calculus preserve goodness of explicit processes, starting from $\{\mathsf{id}\}P_*$ which is indeed good, as $\Phi_{P_*} = \mathsf{tt}$ and $\mathsf{id} \models \mathsf{tt}$.

Lemma 4.6.1 ($\Rrightarrow$ preserves goodness) *For all explicit processes S and S' it holds that*

$$\mathsf{good}(S) \wedge S \Rrightarrow S' \;\Rightarrow\; \mathsf{good}(S')$$

The proof is organised by induction on the shape of the inference tree for the congruence step $S \Rrightarrow S'$.

Lemma 4.6.2 ($\longrightarrow$ preserves goodness) *For all explicit processes S and S' it holds that*

$$\mathsf{good}(S) \wedge S \longrightarrow S' \Rightarrow\; \mathsf{good}(S')$$

The proof is organised by induction on the shape of the inference tree for the transition $S \longrightarrow S'$, exploiting Lemma 4.6.1 for accommodating transitions of congruent processes.

4.7 Concluding Remarks

Many of the errors in current software are due to an overly optimistic programming style. Programmers tend to think of benign application environments and hence focus on getting the software to perform as many functions as possible. To a much lesser extent they consider malign application environments and the need to focus on avoiding errors that can be provoked by outside attackers.

This is confounded by the fact that key software components are often developed in one context and then ported to another. The Simple Mail Transfer Protocol (SMTP) is a case in point. Originally developed in benign research or development environments, where few would be motivated to misuse the protocol and could easily be reprimanded if doing so, it has become a key constituent of the malign environment provided by the global Internet where many users

find an interest in misusing the protocol, and where it is extremely difficult to even identify offenders.

Future programming languages and programming environments need to support a more robust (pessimistic) programming style: what conceivably might go wrong probably will go wrong. A major cause of disruption is due to the communication between distributed software components. Hence, our focus considers how to mitigate the consequences of attacks, nature, or misfortune preventing expected communication from taking place – whatever source of unavailability. This calls for a very robust way of programming systems where there are always default data available for allowing the system to continue its operation as best as it can (rather than simply terminate with an error or get stuck in an input operation).

We believe that the Quality Calculus presents the core ingredients of a process calculus supporting such defensive (robust) programming. To assist in analysing the extent to which robustness has been achieved, we have developed a SAT-based robustness analysis that indicates the places where errors can still arise in spite of robust programming, and where additional hardening of the code may be called for.

The calculus embraces the modelling approach to the development of process algebras. As a matter of fact, a number of papers has already studied process calculi for modelling unreliable communication due to faults in the underlying network [Ama97, RH98, BH00, RH01, Ber04, FH05]. However, previous investigations differ from ours in that they are based on a fairly low-level model of the network topology and of the properties of the nodes, and do not focus on enforcing alternative behaviours when data are unavailable.

From a technical point of view, observe that one of the two intuitions behind the Quality Calculus, i.e., non-blocking binders, had been already proposed in Linda with predicate inputs [ACG86], and then inherited in several versions of KLAIM [BBD+03]. Linda advocated for a programming style where the intention was to check the Boolean result of such an input operation so as to split the continuation process in two branches, and use input variables only in the branch related to a successful input [AAV95, pg. 2-25]. Nonetheless, no enforcement mechanism is provided, while in the Quality Calculus `case` clauses compel to inspect the content of input variables already at the syntactic level, and the robustness analysis completes the picture taking advantage of the new primitives. In the literature on Linda, on the contrary, `inp` and `readp` are studied in relation to their expressiveness with respect to other communication primitives and to the efficiency of their implementations, e.g., in terms of polling (cf. the introduction of [BWA94] and the reference therein). Recently, we have commented on a similar issue in [VCT+14], where security policies might prescribe to skip some actions and thereby the definition of some variables.

Chapter 5

From Network to Application Level

In the previous chapter we have developed a calculus to model network-level DoS, that is, absence of communication, and complemented the formalism with a static analysis that pinpoints where the absence of data might lead to undesirable situations. Building on the same idea, we lift now the approach to reason about application-level DoS: even if the communication takes place, incorrect or corrupted data might be delivered. As a matter of fact, whether we receive nothing or something we cannot use, the *effect* is the same.

In order to encompass both these traits of unavailability, we extend the Quality Calculus with input patterns dictating what sort of messages an input is willing to accept. Such patterns seamlessly integrate with quality binders. A novel SMT-based *availability analysis* takes advantage of the additional information instrumenting the input binders, relating the reachability of program points to the availability of data *with given shape*.

It is worthwhile observing that the availability analysis subsumes the robustness analysis of Ch. 4. Nevertheless, besides fulfilling pedagogical and presentation purposes, there are technical reasons for choosing to present the robustness analysis first, as being propositional it is always decidable, whereas the availability analysis resorts to a first-order encoding where function symbols may range over infinite domains. Even though the analysis we carried out is static, hence a finite bound to these domains can be determined if we restrict to closed systems, still it is more expensive to compute.

The organisation of this chapter follows closely the structure of Ch. 4. First, we extend the calculus with patterns in § 5.1 and present its reduction semantics

Table 5.1: The syntax of the Quality Calculus with patterns.

$$
\begin{array}{rcl}
P & ::= & 0 \mid (\nu c^{\tau})\, P \mid P_1 | P_2 \mid b.P \mid t_1!t_2.P \mid A(e) \\
 & \mid & \mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(p)\colon P_1\ \mathsf{else}\ P_2 \\
b & ::= & t?x[\underline{p}] \mid \&_q(b_1, \ldots, b_n) \\
t & ::= & c \mid c^+ \mid c^- \mid y \mid g(t_1, \ldots, t_n) \\
 & \mid & \mathsf{enc}(t_1, t_2) \mid \mathsf{aenc}(t_1, t_2) \mid \mathsf{sign}(t_1, t_2) \mid \mathsf{hash}(t) \\
e & ::= & x \mid \mathsf{some}(t) \mid \mathsf{none} \mid f(e_1, \ldots, e_n) \\
p & ::= & c \mid c^+ \mid c^- \mid y \mid g(t_1, \ldots, t_n) \mid \_ \mid p\%y \\
 & \mid & \mathsf{enc}(p, t) \mid \mathsf{aenc}(p, t) \mid \mathsf{sign}(p_1, \underline{p_2}) \mid \mathsf{hash}(t) \\
\underline{p} & ::= & c \mid c^+ \mid c^- \mid y \mid g(t_1, \ldots, t_n) \mid \_ \\
 & \mid & \mathsf{enc}(\underline{p}, t) \mid \mathsf{aenc}(\underline{p}, t) \mid \mathsf{sign}(\underline{p}, \underline{p}) \mid \mathsf{hash}(t)
\end{array}
$$

in § 5.2. Then, we revise the WSN example of the previous chapter to take advantage of the enhanced expressiveness (§ 5.3). The availability analysis is developed in § 5.4 and its implementation is discussed in § 5.5, where also some results on the example are reported and commented upon. Finally, the formal statements concerning the correctness of the analysis are presented in Appendix B.

5.1 The Quality Calculus with Patterns

The Quality Calculus with patterns extends the Quality Calculus of Ch. 4 instrumenting binders with patterns for selective input. Moreover, patterns allow to implement cryptographic reasoning with little effort, offering a framework where a novel treatment of availability coexists with the standard approach to confidentiality and authenticity. As far as application-level DoS is concerned, however, cryptographic constructs are just a meaningful example of structured data type, useful to illustrate pattern matching.

The syntax of the extended calculus is displayed in Table 5.1, and includes the constructs of Table 4.1 enhancing some of them with patterns and cryptographic operations.

From network to application level. As we have seen in Ch. 4, the core mechanism that enables to tackle DoS at the network level in the Quality Calculus is the interplay between *data* and *optional data* on one side, and quality binders and case clauses on the other side. For we want now to lift the treatment of unreliable communication from a "received/not received" view to a more

elaborate perspective, where the content of the communication matters, we shall distinguish between *values* and *optional values*.

A term t denotes values: names, variables $y \in \mathcal{Y}$, or function applications. A name is either a plain name c (e.g., a symmetric key), a public key c^+, or a private key c^-. Names are now introduced by the *qualified* restriction $(\nu c^\tau)\, P$, where the qualifier τ is either the empty string ϵ, yielding a standard restriction, or the key pair generator $\pm$ introducing the pair (c^+, c^-) of names in P. Qualified names are inspired by [BRN04], where they have been studied in relation to pattern matching mechanisms. A function application is either a function g from terms to names, or the application of a cryptographic constructor. Constructors $\mathsf{enc}(t_1, t_2)$ and $\mathsf{aenc}(t_1, t_2)$ denote symmetric and asymmetric encryption, respectively, where the first term is the plain-text and the second term the cryptographic key. Similarly, $\mathsf{sign}(t_1, t_2)$ is the signature of term t_1 under the private key t_2, while $\mathsf{hash}(t)$ represents hashing term t. The set $\mathcal{C}$, denoting constant data in Ch. 4, is now extended to range over constant values.

An expression e denotes instead optional values and can be a variable $x \in \mathcal{X}$, the empty expression none denoting no information, the expression $\mathsf{some}(t)$ from values to optional values denoting that e carries an actual value, or a function f from optional values to optional values. The set $\mathcal{O}$, denoting constant optional data in Ch. 4, is now extended to range over constant optional values.

As for binders b, the input $t?x[\underline{p}]$ waits for a message on channel t, and binds x to the message if the latter is matched by the pattern $\underline{p}$, according to the procedure explained below. Simple input binders can be organised in quality binder as in the basic calculus. The process $\&_\exists(t_1?x_1[\underline{p_1}], t_2?x_2[\underline{p_2}]).P$, evolves to P as soon as at least one the input of the quality binder is honoured. As a consequence, it might be that an input variable x_i has not been bound to any message in the continuation process. Lifting the idea of the basic calculus, an input variable always carries optional values: it is bound to $\mathsf{some}(v)$ if v is the values received by the input, or to none if the input is not honoured but we are proceeding anyway.

The construct $\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(p) : P_1\ \mathsf{else}\ P_2$ is then used to check whether an expression (e.g., an input variable) is indeed carrying values, evolving to P_1 if the expression e carries a value matched by the pattern p, or to P_2 if it is none. When the matching is successful a substitution from variables defined in p to terms in corresponding positions of e is generated and applied to the continuation process P_1. It is worthwhile observing that input channels and outputs range over terms, meaning that before using a received optional value we are obliged to extract the value payload (if any) – just as in the basic calculus.

The distinction between values and optional values copes with the potential unreliability of communication, allowing to proceed even when information is not arriving *or not complying with the expected format*, thus addressing the *semantics* of the communication, climbing the Internet protocol suit abstraction stack.

Reasoning with patterns. The usefulness of patterns is two-fold: they are used to select input messages with a given structure, thus attaching "contracts" to inputs, and to implement cryptographic reasoning.

Intuitively, a pattern matches a term if they are syntactically identical. In addition to this, the grammar introduces the wild-card pattern _, which matches everything, and the binding pattern $p\%y$, which binds a term t to the variable y if the pattern p matches t, so that this is a defining occurrence of variable y.

Input patterns allow selecting messages with a given structure. As an example, the input process $t_1?x[\mathsf{enc}(\_, k)].P$ is willing to receive any term (i.e., _) symmetrically encrypted under key k. However, patterns $\underline{p}$ used in input cannot define new variables. This restriction facilitates obtaining an elegant semantics, since quality binders may allow to proceed with the computation even if some inputs have not been honoured. For the sake of discussion, suppose that we were to allow $\underline{p}$ to contain a binding pattern. In the event that an input $t?x[\mathsf{enc}(\_\%y, c)]$ does not arrive, and nonetheless we are allowed to proceed, in the continuation process both x and y should be mapped to none. Syntactically, this would require to replace y with a variable x_1 in $p\%y$, but this is in contrast with the intuition that cryptographic constructors work on values and return values.

As for cryptographic patterns, observe that symmetric and asymmetric encryption require the key to be matched by a term t, symbolically expressing the perfect cryptography assumption: decryption is successful only when the cryptographic key is known (in particular, no wild-card _ is allowed). Similarly, the pattern $\mathsf{hash}(t)$ expresses that a hash can only be compared to another hash, and thus the term on which the hash is built must be known. As for signature, we allow the key to be matched by _ accounting for the possibility that a message is accepted without verifying the signature.

Patterns in case clauses allow binding new variables to matching (sub-)terms. This mechanism plays a central role in inverting encryption constructors, as in

$$t?x[\mathsf{enc}(\_, c)].\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(\mathsf{enc}(\_\%y, c))\colon P_1\ \mathsf{else}\ P_2$$

where the input selects a term encrypted with the key k, and the case clause computes the decryption binding the plain-text to y.

As usual, in the following we shall restrict to closed processes (no free variable), and assume that processes are α-renamed so that variables are defined exactly once. This assumption simplifies the availability analysis without impairing the expressiveness of the framework, as processes are considered *equal* up to α-renaming. Finally, a variable cannot be defined and then applied in the same pattern. This restriction rules out harmful processes like

$$\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(\mathsf{enc}(\_\%y, y))\colon P_1\ \mathsf{else}\ P_2$$

which is semantically incorrect since we cannot obtain the plain-text without knowing the key.

Table 5.2: The evaluation $\triangleright$ of terms into values and of expressions into optional values.

$$c \triangleright c \qquad \frac{t_1 \triangleright v_1 \quad \cdots \quad t_n \triangleright v_n}{g(t_1, \ldots, t_n) \triangleright v} \text{ if } [\![g]\!](v_1, \ldots v_n) = v$$

$$\frac{t_1 \triangleright v_1 \quad t_2 \triangleright v_2}{\mathsf{enc}(t_1, t_2) \triangleright \mathsf{enc}(v_1, v_2)} \qquad \frac{t_1 \triangleright v_1 \quad t_2 \triangleright v_2}{\mathsf{aenc}(t_1, t_2) \triangleright \mathsf{aenc}(v_1, v_2)}$$

$$\frac{t_1 \triangleright v_1 \quad t_2 \triangleright v_2}{\mathsf{sign}(t_1, t_2) \triangleright \mathsf{sign}(v_1, v_2)} \qquad \frac{t \triangleright v}{\mathsf{hash}(t) \triangleright \mathsf{hash}(v)}$$

$$\mathsf{none} \triangleright \mathsf{none} \qquad \frac{t \triangleright v}{\mathsf{some}(t) \triangleright \mathsf{some}(v)}$$

$$\frac{e_1 \triangleright o_1 \quad \cdots \quad e_n \triangleright o_n}{f(e_1, \ldots, e_n) \triangleright o} \text{ if } [\![f]\!](o_1, \ldots o_n) = o$$

5.2 Reduction Semantics

The semantics of the calculus with patterns retraces that of the basic calculus: it consists of a transition relation $P \longrightarrow P'$, parametrised on a structural congruence relation and on some auxiliary relations for terms and expressions evaluation and pattern matching.

Terms and expressions evaluations. The relation $t \triangleright v$ defines how terms are evaluated into *values* v. According to this relation, defined in the first section of Table 5.2, a value is either a name or the application of a cryptographic constructor to values.

$$v ::= c \mid \mathsf{enc}(v_1, v_2) \mid \mathsf{aenc}(v_1, v_2) \mid \mathsf{sign}(v_1, v_2) \mid \mathsf{hash}(v)$$

Hence, the structure of a cryptographic term is maintained by the evaluation, symbolically expressing the possibility of inverting an encryption, checking a signature or a hash, which would not be possible if everything reduced to flat names, as in the basic calculus. As before, $[\![g]\!](\ldots)$ denotes the application of function g to actual parameters.

Similarly, expressions evaluate to optional values, an optional value o being either $\mathsf{some}(v)$ or none. The evaluation $e \triangleright o$ from expressions to optional values is displayed in Table 5.2, second section.

Pattern matching. Pattern matching is defined by the matching relation $\vdash v \bowtie p : \sigma$ between a value v and a pattern p, which produces a substitution σ,

Table 5.3: The value-pattern matching relation $\bowtie$.

$$\vdash c \bowtie c : \mathsf{id} \qquad \vdash v \bowtie \_ : \mathsf{id}$$

$$\vdash c^+ \bowtie c^+ : \mathsf{id} \qquad \vdash c^- \bowtie c^- : \mathsf{id}$$

$$\frac{\vdash v \bowtie p : \sigma}{\vdash v \bowtie p\%y : \sigma[y \mapsto v]} \qquad \frac{\vdash v_1 \bowtie p : \sigma \qquad \vdash v_2 \bowtie v : \mathsf{id}}{\vdash \mathsf{enc}(v_1, v_2) \bowtie \mathsf{enc}(p, v) : \sigma}$$

$$\vdash \mathsf{hash}(v) \bowtie \mathsf{hash}(v) : \mathsf{id}$$

$$\frac{\vdash v_1 \bowtie p_1 : \sigma \qquad \vdash c^- \bowtie v_2 : \mathsf{id}}{\vdash \mathsf{aenc}(v, c^+) \bowtie \mathsf{aenc}(p_1, v_2) : \sigma} \qquad \frac{\vdash v \bowtie p_1 : \sigma \qquad \vdash c^+ \bowtie \underline{p_2} : \mathsf{id}}{\vdash \mathsf{sign}(v, c^-) \bowtie \mathsf{sign}(p_1, \underline{p_2}) : \sigma}$$

that is, a map from variables y to values. The judgements defining the matching relation are presented in Table 5.3. A (qualified) constant c matches itself, and the wild-card pattern matches any value. In these cases no binding takes place, and thus the matching produces the identical substitution. Binding patterns are the only case where the matching produces a non-trivial substitution. As for cryptographic terms, observe that terms for keys are now replaced by values, for our processes are closed and substitutions applied directly, hence when a matching takes place variables are replaced by values (cf. rule (Case-tt) of Table 5.4). As a consequence, pattern matching involving keys yields id. The correspondence of private and public keys is obtained by demanding an asymmetric matching between qualified names in the rule for $\mathsf{aenc}(\cdot, \cdot)$: asymmetric cryptography uses the public key for encryption and the private key for signature, and the corresponding patterns have to match the private and public key, respectively. Finally, observe that a hash is not invertible and thus it can only be compared to another hash. When a value v is not matched by a pattern p, we write $\not\vdash v \bowtie p$, meaning that a substitution which induces the matching cannot be produced ($\nexists \sigma. \vdash v \bowtie p : \sigma$).

Transition relation. As regards the structural congruence, the rules retrace closely those of the congruence for the basic calculus, limiting to replace restrictions with qualified restrictions. Similarly, contexts C now include $(\nu c^\tau)\, C$.

Table 5.4 defines the transition relation, extending the reduction semantics of the Quality Calculus by integrating pattern matching at the input and case level. The first part of the table displays the transition rules, the second part shows the relation between outputs and binders, and the the third part updates the Boolean evaluation of binders.

The meaning of rules (In-ff) and (In-tt) is left unchanged. Three rules are now needed to accommodate the treatment of $\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(p) : P_1\ \mathsf{else}\ P_2$. If

e evaluates to $\mathsf{some}(v)$ and v is matched by pattern p, then the test is successful and the substitution σ produced by the matching is applied to P_1, as dictated by rule (Match); otherwise, if v is not matched by p – rule (Mismatch)– or the expression evaluates to none – rule (Case-ff) –, then the else branch is followed.

It is worthwhile noticing that substitutions σ differ from substitutions θ, in that the latter map variables $x \in \mathcal{X}$ to optional values $o \in \mathcal{O}$, as they originate from binders. When a case clause or an input is successful, the related substitution is applied to the continuation process. Therefore, as we consider closed processes, whenever a term t is matched against a pattern p in an actual execution, all the variables in t have already been replaced with values. This behaviour justifies the restriction to values in the definition of the pattern matching relation of Table 5.3. Likewise, when a matching is evaluated, all applied occurrences of variables in the pattern have been replaced with constants.

Finally, like in the basic calculus, we have a rule for the unfolding of recursive calls, a rule linking the transition relation to the structural congruence, and a rule for transition in contexts, which takes care of interleaving and leading restrictions.

The relation $c!v \vdash b \to b'$ describes the effect of an output on a binder, whose syntax is extended to include simple substitutions like $[\mathsf{some}(v)/x]$, denoting a satisfied input. Given an output and an input synchronising on the same channel, if the input pattern matches the received *value* v, then the input variable is bound to the *optional value* $\mathsf{some}(v)$. The matching yields no substitution, as input patterns cannot define variables. This behaviour is seamlessly embedded into quality binders. Observe that the output term is allowed to be a value v, while we still require the channel to be a name (cf. the rules for input).

$b ::_r \theta$ establishes whether binder b is sufficiently instantiated ($r = \mathsf{tt}$) or needs more input ($r = \mathsf{ff}$), recording in θ the bindings of input variables. With respect to the corresponding relation of Table 4.4 we need only to update the syntax, accounting for patterns and replacing optional data with optional values.

5.3 The Base Station, Revised

Let us revise the example of § 4.4 in a more security-oriented mind-set. Consider a Home Area Network where a smart meter SM (base station) is in charge of scheduling connected appliances in order to optimise the energy budget [VYD12]. As the network includes a domestic photovoltaic system, SM relies on an external wireless sensor WF (sensor node) for weather forecasting, so as to estimate the amount of energy that will be produced in-house. Due to the insecurity of the wireless medium, the communication between WF and SM is symmetrically encrypted. Moreover, due to the unreliability of the connection and of the sensor itself, which is placed outdoor, SM relies on a local computer to provide an estimate based on historical data whenever WF is not responding in due time, as dictated by a local clock.

Table 5.4: The reduction relation $\longrightarrow$ of the Quality Calculus with patterns.

$$\frac{t_1 \rhd c_1 \quad t_2 \rhd v_2 \quad c_1!v_2 \vdash b \to b' \quad b' ::_{\mathsf{ff}} \theta}{t_1!t_2.P_1|b.P_2 \longrightarrow P_1|b'.P_2} \text{(In-ff)}$$

$$\frac{t_1 \rhd c_1 \quad t_2 \rhd v_2 \quad c_1!v_2 \vdash b \to b' \quad b' ::_{\mathsf{tt}} \theta}{t_1!t_2.P_1|b.P_2 \longrightarrow P_1|P_2\theta} \text{(In-tt)}$$

$$\frac{e \rhd \mathsf{some}(v) \qquad \vdash v \bowtie p : \sigma}{\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(p)\colon P_1\ \mathsf{else}\ P_2 \longrightarrow P_1\sigma} \text{(Match)}$$

$$\frac{e \rhd \mathsf{some}(v) \qquad \not\vdash v \bowtie p}{\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(p)\colon P_1\ \mathsf{else}\ P_2 \longrightarrow P_2} \text{(Mismatch)}$$

$$\frac{e \rhd \mathsf{none}}{\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(p)\colon P_1\ \mathsf{else}\ P_2 \longrightarrow P_2} \text{(Case-ff)}$$

$$\frac{}{A(e) \longrightarrow P[e/x]} \text{ if } A(x) \triangleq P \text{(Rec)}$$

$$\frac{P_1 \equiv P_2 \quad P_2 \longrightarrow P_3 \quad P_3 \equiv P_4}{P_1 \longrightarrow P_4} \text{(Cng)} \qquad \frac{P_1 \longrightarrow P_2}{C[P_1] \longrightarrow C[P_2]} \text{(Cnt)}$$

$$\frac{t \rhd c \qquad \vdash v \bowtie p : id}{c!v \vdash t?x[\underline{p}] \to [\mathsf{some}(v)/x]}$$

$$\frac{c_1!c_2 \vdash b_i \to b_i'}{c_1!c_2 \vdash \&_q(b_1, \cdots, b_i, \cdots, b_n) \to \&_q(b_1, \cdots, b_i', \cdots, b_n)}$$

$$t?x[\underline{p}] ::_{\mathsf{ff}} [\mathsf{none}/x] \qquad [\mathsf{some}(v)/x] ::_{\mathsf{tt}} [\mathsf{some}(v)/x]$$

$$\frac{b_1 ::_{r_1} \theta_1 \quad \cdots \quad b_n ::_{r_n} \theta_n}{\&_q(b_1, \cdots, b_n) ::_r \theta_n \cdots \theta_1} \text{ where } r = [\![q]\!](r_1, \cdots, r_n)$$

In the following formalisation we shall use some operators derived from quality binders as shown in § 4.3; the extension to the syntax with patterns is straightforward.

The sensor is specified in the calculus as the following process:

$$\begin{aligned}\mathsf{WF} \triangleq\ & 0 \oplus (\mathsf{sm}?x[\mathsf{enc}(\_, k)]).\\ & \mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(\mathsf{enc}(\_\%y_m, k)) : \mathsf{sm}!\mathsf{enc}(\mathsf{estimate}(y_m), k).\mathsf{WF}\ \mathsf{else}\ 0.\mathsf{WF}\end{aligned}$$

The top-level choice specifies that WF either dies (physical attack, depleted battery, ...) or operates as expected. In the ideal case, the sensor waits for a forecast request from SM on channel sm, encrypted under a shared key k and containing the time-point to forecast. If such a message is received, then the plain-text is extracted in y_m and the estimate $\mathsf{estimate}(y_m)$ (a function g) is encrypted and returned. Observe that there is no need for testing whether or not the content of the encrypted term is other than none, as an encryption can only be built on values.

As for the smart meter, we obtain the following process:

$$\begin{aligned}\mathsf{SM} \triangleq\ & (\nu m)\,(\nu t)\,\&^{!}_{\exists}(\mathsf{lc}!m, \mathsf{sm}!\mathsf{enc}(m, k)).\mathsf{set}!t.\\ & \&_{\forall}(\mathsf{tick}?x_t[\_], \&^{?}_{\exists}(\mathsf{lc}?x_l[\_], \mathsf{sm}?x_r[\mathsf{enc}(\_, k)])).\\ & \mathsf{case}\ x_r\ \mathsf{of}\ \mathsf{some}(\mathsf{enc}(\_\%y_r, k)) : {}^{1}\mathsf{store}!y_r.\mathsf{SM}\ \mathsf{else}\\ & \qquad \mathsf{case}\ x_l\ \mathsf{of}\ \mathsf{some}(\_\%y_l) : {}^{2}\mathsf{store}!y_l.\mathsf{SM}\ \mathsf{else}\ {}^{3}0\end{aligned}$$

First, the smart meter sends the request for a new measure m to the local computer on channel lc and to the sensor node. When at least one request is accepted, SM sets a deadline t starting the local clock, and then starts waiting. The quality binder in the second line is consumed as soon as (i) the deadline has expired (input from the clock on channel tick), and (ii) either the local computer or the sensor node has responded. After the binder is consumed the smart meter extracts the received value and stores it locally: in case WF replied, then the message has to be decrypted, otherwise if the local computer responded the value is assumed to be in clear.

It is worthwhile observing that when the binder in the second line of SM is consumed, it must be the case that either the local estimate or the more reliable one computed by the remote sensor is available, the implicit assumption being that local components will always respond. In other words, the program points instrumented with labels $1, 2$ must be reachable, and they are so under a precise characterisation for the expected inputs, while label 3 is not. We will show in § 5.4 how the analysis confirms such intuition.

The processes defining the clock and the local computer essentially retrace those of § 4.4 to the smart meter requests and thus we omit them for the sake of brevity, as the analysis will only concern SM.

5.4 Availability of Communication

The robustness analysis of Ch. 4 is limited to consider whether or not an input is received, whereas patterns enable to inspect the structure of messages, considering *what* is to be received. The language of § 5.1 allows indeed to record whether or not a given binder has been satisfied, as well as the optional values its variables may assume according to the input patterns, and thus yields a more precise analysis.

5.4.1 Availability analysis

The availability analysis, defined in Table 5.5 on the structure of processes, is based on the judgement

$$\vdash \xi \mathbin{@} P$$

denoting that the first-order formula ξ describes the reachability condition of the program point just before P, the idea being that if P is reachable then ξ is satisfiable and, by contra-position, *if ξ is unsatisfiable then P is unreachable.* We assume that the occurrence of P in the main process P_* can be uniquely identified – labels or tree addresses can be used to solve ambiguities, but in the following we shall dispense with them to simplify the notation. Moreover, we assume that all the variables are existentially quantified at the outermost level: scope confusion cannot occur thanks to the assumption that processes are α-renamed. Though this choice yields more readable formulae, clearly superior for presentation purposes, an efficient implementation would introduce explicit existential quantifiers instead of using fresh constants, letting the SMT solver perform Skolemisation.

Our first-order formulae ξ use the uninterpreted function symbol $\mathsf{some}(\cdot)$ and the constant none to model optional values, and relate expressions to the optional values they carry by means of the equality symbol, as in $x = \mathsf{some}(c)$.

In the first part of Table 5.5 we assume that each process defined in the system may be executed. Then, in the second part, rules (A-new), (A-par1), (A-par2), (A-out) express that restriction, parallel composition, and output do not require additional information to proceed with the computation nor impose shape constraints on information received so far.

Binders and case constructs determine instead the shape of ξ, as they rule the content of accepted messages and account for their availability. Rule (A-bin) abstracts the behaviour of binders resorting to the judgement

$$\vdash b \blacktriangleright \varphi, \psi$$

defined in the last section of Table 5.5. A binder is modelled by two formulae:

- φ, recording the combination of inputs that allow passing the binder (success condition); and,

Table 5.5: The availability analysis $\vdash \xi \mathbin{@} P$ of the Quality calculus with patterns.

$$\vdash \mathsf{tt} \mathbin{@} P_* \qquad \vdash \mathsf{tt} \mathbin{@} P_1 \qquad \cdots \qquad \vdash \mathsf{tt} \mathbin{@} P_n$$

$$\frac{\vdash \xi \mathbin{@} (\nu c)\, P}{\vdash \xi \mathbin{@} P}\ \text{(A-new)} \qquad \frac{\vdash \xi \mathbin{@} (P_1 \mid P_2)}{\vdash \xi \mathbin{@} P_1}\ \text{(A-par1)} \qquad \frac{\vdash \xi \mathbin{@} (P_1 \mid P_2)}{\vdash \xi \mathbin{@} P_2}\ \text{(A-par2)}$$

$$\frac{\vdash \xi \mathbin{@} (t_1!t_2.P)}{\vdash \xi \mathbin{@} P}\ \text{(A-out)} \qquad \frac{\vdash \xi \mathbin{@} b.P \quad \vdash b \blacktriangleright \varphi_b, \psi_b}{\vdash \xi \wedge \varphi_b \wedge \psi_b \mathbin{@} P}\ \text{(A-bin)}$$

$$\frac{\vdash \xi \mathbin{@} (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(p) \colon P_1\ \mathsf{else}\ P_2) \quad \vdash e \bowtie p : \psi_e}{\vdash \xi \wedge \psi_e \mathbin{@} P_1}\ \text{(A-case)}$$

$$\frac{\vdash \xi \mathbin{@} (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(p) \colon P_1\ \mathsf{else}\ P_2) \quad \vdash e \bowtie p : \psi_e}{\vdash \xi \wedge \neg(\exists \mathsf{bv}(p).\psi_e) \mathbin{@} P_2}\ \text{(A-else)}$$

$$\frac{\vdash x \bowtie \underline{p} : \psi_x}{\vdash t?x[\underline{p}] \blacktriangleright x = \mathsf{some}(y), (\psi_x \vee x = \mathsf{none})}$$

$$\frac{\vdash b_1 \blacktriangleright \varphi_1, \psi_1 \quad \cdots \quad \vdash b_n \blacktriangleright \varphi_n, \psi_n}{\vdash \&_q(b_1, \ldots, b_n) \blacktriangleright [\![q]\!](\varphi_1, \ldots, \varphi_n), (\psi_1 \wedge \cdots \wedge \psi_n)}$$

- ψ, which describes the optional values that the input variables may assume according to the input patterns (strongest post-condition).

Consider a simple input $t?x[p]$. The success condition φ simply records that x has to be received in order to pass the input, disregarding the received optional value, which can be built on any y, where $y \in \mathcal{Y}$ is a fresh variable that does not appear elsewhere in the formulae generated so far. The strongest post-condition ψ is a disjunction of two sub-formulae: ψ_x, which records the structure of the optional value to which a successful input binds x, and $x = \mathsf{none}$, recording the possibility that the input is not performed. The component ψ_x of the strongest post-condition is produced according to a judgement for pattern matching, explained below.

Quality binders build on the same intuition: the success condition is obtained by applying the quality guard q to the success condition of the sub-binders, as much as in the robustness analysis of § 4.6.1; the strongest-post condition is instead obtained as the conjunction of the strongest post-conditions of the sub-binders.

The analysis of case constructs relies on a judgement for pattern matching,

as it is the case for input binders. In the successful branch of the test, rule (A-case), it must be that the expression e complies with the shape dictated by the pattern p, described by a strongest post-condition formula ψ_e. As for a failing case test, rule (A-else), we state the negative knowledge we have acquired on the shape of e by negating an existential quantification over the variables bound in p: there is no value for $\mathsf{bv}(p)$ such that the matching is successful.

The judgement $\vdash e \bowtie p : \psi$ is presented in Table 5.6, and produces the strongest post-condition generated by a successful pattern matching, that is, a formula that describes the *shape* with which an expression e has to comply in order to match a pattern p. Ideally, this is achieved by imposing $e = \mathsf{some}(p)$, but due care has to be paid to wild-cards and defining occurrences of variables appearing in p.

Before computing the analysis, we replace each occurrence of the wild card _ with a fresh variable $y^{\underline{i}} \in \mathcal{Y}$. For defining occurrences of variables do not contribute to describe the shape of e, we get rid of them applying $\mathsf{strip}(p)$, which produces a copy of p without variable definitions. Nonetheless, as the definition of a variable y may carry information about the structure of the term to which y is bound, a conjunct recording every definition is generated by $\mathsf{exp}(p)$. Observe that the definition of $\mathsf{strip}(\cdot)$ and $\mathsf{exp}(\cdot)$ on cryptographic constructors takes advantage of the perfect cryptography assumption: patterns in position of keys or hashed terms cannot bind variables. Finally, observe that patterns $\underline{p}$ are a subset of patterns p, hence the definition of the matching judgement is straightforward.

As an example, consider the clause

$$\mathsf{case}\ x_r\ \mathsf{of}\ \mathsf{some}(\mathsf{enc}(y^{\underline{r}}\%y_r, k)) \colon \ldots$$

of process SM, where $y^{\underline{r}}$ is a fresh variable replacing the wild-card. The formula generated by the analysis to describe the strongest post-condition in the successful branch of the test is

$$\vdash x_r \bowtie \mathsf{enc}(y^{\underline{r}}\%y_r, k) : \big(x_r = \mathsf{some}(\mathsf{enc}(y^{\underline{r}}, k)) \wedge \mathsf{tt} \wedge (y_r = y^{\underline{r}})\big)$$

All the variables are assumed to be existentially quantified. The first conjunct records the shape that x_r must have in order for the flow to pass the test successfully, that is, the variable must carry an encryption under key k. The second conjunct is obtained by $\mathsf{exp}(y^{\underline{r}})$; this is useful in case nested patterns occur that define the format of a bound variable, but in this case can be safely ignored. The third conjunct imposes that variable y_r, defined in the pattern, has the format of the term matched by the pattern; being such term another fresh variable, also this conjunct can be safely ignored. As for the formula describing a failed matching, we obtain

$$\neg\exists y^{\underline{r}}\,(x_r = \mathsf{some}(\mathsf{enc}(y^{\underline{r}}, k)))$$

stating that whatever value x_r is carrying, if any, it is not an encryption under key k (trivial conjuncts are omitted).

Table 5.6: The matching judgement $\vdash e \bowtie p : \psi$.

$$\vdash e \bowtie p : \big(e = \mathsf{some}(\mathsf{strip}(p)) \wedge \mathsf{exp}(p)\big)$$

$$\mathsf{strip}(c) = c \qquad \mathsf{strip}(y) = y \qquad \mathsf{strip}(p\%y) = \mathsf{strip}(p)$$
$$\mathsf{strip}(\mathsf{enc}(p_1, p_2)) = \mathsf{enc}(\mathsf{strip}(p_1), p_2) \qquad \mathsf{strip}(\mathsf{aenc}(p_1, p_2)) = \mathsf{aenc}(\mathsf{strip}(p_1), p_2)$$
$$\mathsf{strip}(\mathsf{sign}(p_1, p_2)) = \mathsf{sign}(\mathsf{strip}(p_1), p_2) \qquad \mathsf{strip}(\mathsf{hash}(p)) = \mathsf{hash}(p)$$

$$\mathsf{exp}(c) = \mathsf{tt} \qquad \mathsf{exp}(y) = \mathsf{tt} \qquad \mathsf{exp}(p\%y) = \mathsf{exp}(p) \wedge (y = \mathsf{strip}(p))$$
$$\mathsf{exp}(\mathsf{enc}(p, t)) = \mathsf{exp}(p) \qquad \mathsf{exp}(\mathsf{aenc}(p, t)) = \mathsf{exp}(p)$$
$$\mathsf{exp}(\mathsf{sign}(p_1, p_2)) = \mathsf{exp}(p_1) \qquad \mathsf{exp}(\mathsf{hash}(t)) = \mathsf{tt}$$

5.4.2 Formal correctness

The correctness statement is phrased as follows:

$$\big(\vdash \xi \mathbin{@} P \;\wedge\; (P_* \longrightarrow^* C[P\vartheta])\big) \;\Rightarrow\; \big(\vartheta \models \xi\big)$$

that is, if ξ describes the reachability of P in a system and P_* reaches a context where P is instantiated under substitution ϑ for input variables and variables defined by patterns, then ϑ is a model for ξ. The full-fledged formal proof resorts to a semantics with explicit substitution and mimics closely the correctness proof of the robustness analysis. Appendix B.1 contains the main statements that are needed in order to lift the proof of the robustness analysis to the availability analysis.

5.5 Implementation of the Analysis

In the following, we show how to encode the analysis as an SMT problem in SMT-LIB [BST10], using Z3 to decide the satisfiability of the resulting formulae and demonstrating the procedure on the example of § 5.3.

5.5.1 SMT-LIB encoding

The theory with respect to which we seek satisfiability is given by the combination of first-order logic and the theory of equality with uninterpreted function symbols, together with some first-order axioms described below. The theory makes use of the types Value and OpValue, for values and optional values, respectively. Moreover, we define the function symbols $\mathsf{enc} : \mathsf{Value} \times \mathsf{Value} \to \mathsf{Value}$

(same sort for $\mathsf{aenc}(\cdot,\cdot), \mathsf{sign}(\cdot,\cdot)$), $\mathsf{hash} : \mathsf{Value} \to \mathsf{Value}$, $\mathsf{some} : \mathsf{Value} \to \mathsf{OpValue}$, and the constant none of type $\mathsf{OpValue}$.

When computing a model, Z3 is free to choose a definition for uninterpreted function symbols (in our case the cryptographic constructors and some). Nonetheless, we need to limit the *confusion* allowed in a model, for example requiring that encryptions computed on different values produce distinct results. From the theory of equality with uninterpreted functions we inherit that the functions we have defined indeed behave like mathematical functions (congruence property). Additionally, we require that these functions are injective, that in the case of symmetric encryption translates into

$$\forall y_1, y_2, y_3, y_4 : \mathsf{Value} \;.\; (\mathsf{enc}(y_1, y_2) = \mathsf{enc}(y_3, y_4)) \Rightarrow (y_1 = y_3 \;\wedge\; y_2 = y_4) \quad (5.1)$$

thus ensuring that $\mathsf{enc}(c_1, c) \neq \mathsf{enc}(c_2, c)$ whenever $c_1 \neq c_2$ (and similarly for the other functions). Moreover, we need to distinguish between optional values computed by some and the optional value assigned to the constant none, thus we state:

$$\forall y : \mathsf{Value} \;.\; \mathsf{some}(y) \neq \mathsf{none}$$

Analogously, all the constants in the system under analysis must be kept distinct, but in our example we have only the constant k.[1]

It is worth noticing that enc requires two arguments of type Value as input and produces Value as output. As a consequence, the injectivity axiom (5.1) can only be satisfied if Value contains one element or it is infinite. Since the presence of more than one element in Value will usually be required in meaningful applications (e.g., inputs receiving distinct optional values), it follows that Value must be infinite. Hence, the injectivity axiom can only be satisfied by an infinite model, leading to non-termination in the satisfiable case (jeopardising the decidability result for the satisfiability of term algebras via quantifier elimination). In order to overcome this limitation, we restrict the implementation to consider a fixed number $v_1, \ldots, v_n$ of elements of type Value, and we model a finite fragment of the term algebra where elements can be uniquely decomposed. The restriction is achieved thanks to the introduction of a predicate $\mathsf{dom} : \mathsf{Value} \to \mathsf{Bool}$, such that

$$\forall y : \mathsf{Value} \;.\; \left(\mathsf{dom}(y) \Leftrightarrow \left(\bigvee_{i=1}^{n} (y = v_i)\right)\right)$$

This cardinality constraint is then enforced by requiring that each variable y occurring in a formula generated by the analysis is in the domain induced by

[1] Observe that these axioms would not be needed if the analysis were implemented via resolution-based theorem proving, as we would operate on the initial algebra (Herbrand universe). Refer to [JW09] for a brief comparison of the two approaches.

dom. The injectivity axiom is thus relaxed as follows:

$$\forall y_1, y_2, y_3, y_4 : \mathsf{Value}\ .\\ \left(\bigwedge_{i=1}^{4} \mathsf{dom}(y_i)\right) \wedge (\mathsf{enc}(y_1, y_2) = \mathsf{enc}(y_3, y_4)) \Rightarrow (y_1 = y_3\ \wedge\ y_2 = y_4)$$

This is an effective choice as we analyse finite processes exchanging finite-depth terms: n can be statically over-approximated by looking at the number of variables y and at the maximum depth of an encrypted value.[2] Furthermore, we also need to state that the encryption function enc computes within the domain, and analogously for the other functions.

Finally, it is reasonable to require that the codomains of different functions are mutually disjoint:

$$\forall y_1, y_2, y_3, y_4, y_5, y_6, y_7 : \mathsf{Value}\\ \mathsf{distinct}(\mathsf{enc}(y_1, y_2), \mathsf{aenc}(y_3, y_4), \mathsf{sign}(y_5, y_6), \mathsf{hash}(y_7))$$

so as to distinguish between different objects.

5.5.2 Analysing the smart meter

Let us now apply the analysis to our example. According to the rules, we start with $\vdash \mathsf{tt} \,@\, \mathsf{SM}$. The program point after the generalised output prefix $\&^{!}_{\exists}(\mathsf{lc}!m, \mathsf{sm}!\mathsf{enc}(m, k))$ is described by

$$\begin{aligned}\xi_1 = \quad & (x_1 = \mathsf{some}(y_1) \vee\ x_2 = \mathsf{some}(y_2)) \wedge\\ & (x_1 = \mathsf{some}(y^{\underline{1}}) \vee x_1 = \mathsf{none}) \wedge\\ & (x_2 = \mathsf{some}(y^{\underline{2}}) \vee x_2 = \mathsf{none})\end{aligned}$$

where the first line expresses the success condition imposed by quality guard, asking that at least one input acknowledging an output is received, while the second and third lines consist in the conjunction of the strongest post-conditions of the two inputs into which the output prefix is transformed. Similarly, the analysis of the quality binder

$$\&_{\forall}(\mathsf{tick}?x_t[\_], \&^{?}_{\exists}(\mathsf{lc}?x_l[\_], \mathsf{sm}?x_r[\mathsf{enc}(\_, k)]))$$

produces the formula

$$\begin{aligned}\xi_2 = \quad & (x_t = \mathsf{some}(y_3) \wedge (x_l = \mathsf{some}(y_4) \vee x_r = \mathsf{some}(y_5))) \wedge\\ & (x_t = \mathsf{some}(y^{\underline{t}}) \vee x_t = \mathsf{none}) \wedge\\ & (x_l = \mathsf{some}(y^{\underline{l}}) \vee x_l = \mathsf{none}) \wedge\\ & (x_r = \mathsf{some}(\mathsf{enc}(y^{\underline{r}}, k)) \vee x_r = \mathsf{none})\end{aligned}$$

where the first line expresses the success condition as determined by the nested quality binders: the deadline has to expire (x_t is received and thus bound to some

[2] An alternative approach consists in limiting the depth of nested encryptions to a given bound explicitly, perhaps using a more elaborate type system.

value), and either the local computer or the remote sensor has to respond (x_l or x_r is bound to some value, or both). The last three lines describe the strongest post-conditions of three inputs, showing for instance that if x_r is received, then it must carry an encryption under k. Finally, at the three labels we obtain the formulae

$$\begin{aligned}
1: &\quad \xi_1 \wedge \xi_2 \wedge (x_r = \mathsf{some}(\mathsf{enc}(y^{\underline{r}}, k))) \\
2: &\quad \xi_1 \wedge \xi_2 \wedge \neg\exists y^{\underline{r}}\,(x_r = \mathsf{some}(\mathsf{enc}(y^{\underline{r}}, k))) \wedge (x_l = \mathsf{some}(y^{\underline{l}})) \\
3: &\quad \xi_1 \wedge \xi_2 \wedge \neg\exists y^{\underline{r}}\,(x_r = \mathsf{some}(\mathsf{enc}(y^{\underline{r}}, k))) \wedge \neg\exists y^{\underline{l}}(x_l = \mathsf{some}(y^{\underline{l}}))
\end{aligned}$$

For the sake of readability we have omitted tt conjuncts and conjuncts of the form $(y_r = y^{\underline{r}})$ generated by input of the form $t?x[\_]$ and binding patterns such as $\_\%y_r$, for they do not add any information about the variable y carried by the optional value to which x is possibly bound. At label 1 we have passed the binders $(\xi_1 \wedge \xi_2)$, x_r must be available and carry an encryption. At label 2 instead x_r is not available – either because it nothing has been received at all or nothing of the expected shape – but x_l is carrying some value. Finally, at label 3 we have passed the binders but neither x_r nor x_l is available, which leads to a contradiction.

5.5.3 Results

When studying a process, we need to feed the SMT solver with the formulae generated by the analysis, instrumented with the domain restriction, where all the input variables are declared as constants of type OpValue and all the variables y are introduced with type Value. The complete SMT-LIB code of the motivating example is available at www.imm.dtu.dk/~rvig/quality-smt.z3 (observe that the implementation contains also some redundant conjuncts we have omitted in the discussion above).

Running Z3 on our example (with the option -smt2 for the SMT2 input format), we obtain that labels 1 and 2 *may* be reachable (the formulae are satisfiable) while label 3 is not (the formula is unsatisfiable). The conditions under which labels 1 and 2 may be reachable are derived from the definition that Z3 provides for $\mathsf{enc}(\cdot,\cdot)$, $\mathsf{some}(\cdot)$, and by the assignment computed for the input variables, which can take the optional value none or optional values in the codomain of $\mathsf{some}(\cdot)$. Studying label 2, for example, we have that $x_r = \mathsf{none}$, stating that x_r is not available, while $x_l = \mathsf{some}(\cdot)$: we are in the case in which only the local estimate is available. Moreover, if we were to force that x_r is other than none, we would realise that label 2 would not be reachable any more, for the branching structure of SM would lead to label 1. Finally, the analysis provides us with information about the structure of the messages: at label 1, for instance, we have $x_r = \mathsf{some}(\mathsf{enc}(y_r, k))$, highlighting that x_r is received and carries an encryption. Again, by stating that x_r carries something else than an

encryption under k, we would realise that label 1 would not be reachable any more.

It is worthwhile observing that an under-estimate of the cardinality of the domain dom may result in unsatisfiability, hence due care is required by this step. In particular, any change to the model demands to re-compute the size of the domain.

The SMT-LIB implementation models the analysis as faithfully as possible. In practice, it is possible to use a different representation of the problem in order to obtain remarkably better performance. First of all, it is possible to introduce a new type Key and to re-define cryptographic functions as working on keys, like for instance in $\mathsf{enc} : \mathsf{Value} \times \mathsf{Key} \to \mathsf{Value}$. Since the number of keys is in general much smaller than the number of exchanged messages, this choice considerably improves scalability (i.e., reduces the cardinality of domain). In addition, this approach mimics modern cryptosystems, where keys must fulfil precise conditions, and thus they are distinguishable from other objects. Secondly, introducing existential quantifiers in formulae and letting Z3 perform the Skolemisation step also improves the performance.

Our example set is not sufficient for an extensive discussion of performance issues, but still it is interesting to note that the original formalisation of the example discussed above leads to verify the reachability of label 2 in about 3 minutes when the domain contains 6 elements (on an ordinary laptop), while the improved version (dedicated type for keys and no preliminary Skolemisation) takes about 0.12 seconds. Both the improvements seem to yield a significant gain. The implementation is available at

www.imm.dtu.dk/~rvig/quality-smt-efficient.z3

5.6 Concluding Remarks

We have now concluded our paradigmatic treatment of unavailability, which led to factoring the sources of DoS into unreliable communication and improper information and devise a process calculus where these concepts are first-class citizens. This characterisation can be justified with reference to the layers of the Internet protocol suit, and is implemented by means of programming abstractions that address atomically the behaviours of interest. Levering such new primitives, we propose a static analysis that uncovers where and why DoS may occur, fostering an availability-oriented mind-set from the very first stage of the design.

The centrality of the semantic unavailability treated in this chapter is corroborated by the SYN cookies technique discussed in § 3.1.3, which avoids SYN flooding exactly because packets with incorrect format are discarded without further processing. Due to the non-blocking nature of output actions over TCP, however, the formalisation of that example requires a broadcast version of the

calculus like those presented in the following.

The exploitation of patterns for expressing cryptographic reasoning is common in the literature on security protocols, also in those frameworks that rely mainly on rewriting systems. Our usage of patterns is in the wake of [BRN04], where they are used in association with input actions. In this sense, the novelty of our developments consists in the interpretation of input contracts for determining availability condition imposed by the semantics of an application.

Chapter 6

A Broadcast Scenario

In Chs. 4,5 we have presented our paradigmatic approach to enforcing availability, focusing on the primitives that allow modelling DoS at the transport and application level. We shall now turn our attention to applied scenarios, where unavailability threats coexist with broadcast communication and security demands.

In this chapter we present a calculus of broadcasting processes, the Applied Quality Calculus (AQC), instrumented with a theory that allows modelling and reasoning about cryptographic primitives, and equipped with explicit notions of communication failure and unwanted communication, encompassing the developments of Ch. 5. The calculus is to be understood as a modelling effort on which the subsequent developments are based.

With respect to the calculus of Ch. 5, the AQC is equipped with a single syntactic category for messages, and distinguish between value and optional values by means of a simple type system. On the semantic side, the AQC relies on an asynchronous instant communication model, where a process is always allowed to perform an output (broadcast) and continue, while an input is allowed only when a matching output is being performed. Similarly to the Quality Calculus with patterns, inputs are parametrised so as to accept only messages with specific properties (e.g. format), ignoring unwanted communication and thus cutting down the confusion generated by broadcasting over a few (often just one) wireless channels.

Equational reasoning is implemented in the AQC by means of term rewriting, and leveraged for modelling both selective inputs and cryptographic operations. A rewrite theory for cryptographic primitives is presented, which relies on a

simple yet powerful approach for defining cryptographic material, closer to real cryptosystems than other signature-based calculi, without the burden of an explicit type system. Furthermore, conditional rewrite rules allow to specify more complex quality guards, which can relate the behaviour of distinct quality binders.

Finally, the modelling expressiveness of the framework is illustrated on a meaningful case study, where two nodes of a wireless sensor network perform a key update exploiting asymmetric cryptography and secret sharing in order to hamper the work of an attacker. The example highlights how the calculus facilitates dealing with denial-of-service (expected information is not received), flooding generated by broadcast communication (receiving messages non-pertinent to the protocol), and cryptographic reasoning at the same time.

The chapter first presents the AQC, focusing on its syntactic novelties in § 6.1, on rewrite rules in § 6.2, and on the broadcast semantics in § 6.3. The expressive power of the calculus is demonstrated on the example of § 6.4. An account of related works both in the area of broadcasting calculi and in the area of equational reasoning is presented in § 6.5.

This chapter is based on [VNR13].

6.1 The Applied Quality Calculus

The AQC is a statically-typed process calculus. According to the syntax displayed in Table 6.1, a process P consists of actions that range over expressions. An expression e can be a variable x, a name c, or a function application $f(e_1, \ldots, e_n)$. The syntax assumes to have a sorted signature Σ containing elements of the form $(f : s_f)$, where f is a function symbol and $s_f = t_1 \times \cdots \times t_n \to t$ is its sort, defining f as a function of arity n, the types $t_1, \ldots, t_n$ of its arguments, and the type t of the expression computed by f. Constants are represented as functions of arity 0.

Values and optional values. The AQC insists on the distinction between *values* and *optional values* peculiar to Quality Calculi. This distinction is here formalised by means of a simple *type system*: the type V identifies expressions that convey information (values), whereas the type $V^?$ identifies expressions that possibly do not carry information (optional values). A name c has type V, a function f is typed according to its sort, none has type $V^?$, while $\mathsf{some} : V \to V^?$ takes an expression of type V and returns an expression of type $V^?$. A variable x could have either type V or $V^?$; for the sake of consistency with previous chapters, in the following we shall write x to denote variables of type $V^?$ and y for variables of type V. Similarly, we use c to denote names (type V), v to denote values (type V), and o to denote optional values (type $V^?$).

Table 6.1: The syntax of the Applied Quality Calculus.

P	::=	$(\nu \overrightarrow{c}; W)P \mid P_1 \mid P_2 \mid 0 \mid b.P \mid e_1!e_2.P \mid A(e)$
	$\mid$	$\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2$
b	::=	$e_1?x[e_2] \mid \&_{f(e_1,\ldots,e_m)}(b_1,\ldots,b_n)$
e	::=	$x \mid c \mid f(e_1,\ldots,e_n) \mid \mathsf{none} \mid \mathsf{some}(e)$

Equational reasoning. The behaviour of a function application is defined by a set $\mathcal{F}$ of conditional rewrite rules of the form

$$f(e_1,\ldots,e_n) \to e \text{ if } cond$$

where f and all the function symbols occurring in the e_i's belong to Σ, the e_i's contain all the variables occurring in the result e and in the side condition *cond*. Valid constraints in the condition are limited to testing whether or not a list of names is in a given relation, e.g., whether or not two names form an asymmetric key pair, or checking whether a parameter e evaluates to an expression e', according to the following syntax:

$$cond ::= \overrightarrow{c} \in \mathcal{R} \mid e \rhd e' \mid cond \wedge cond \mid cond \vee cond \mid \neg cond \mid \exists z.cond$$

where z has either type D or $D^?$ in $\exists x.cond$. The keyword otherwise is a shorthand notation used in place of if *cond* to denote a rule which applies when no other rule can be applied for the given function symbol. A rewrite theory containing otherwise can always be re-rewritten into an equivalent theory without such keyword [CDE+11, § 4.5.4]. We assume that e and the e_i's are typed coherently with the sort of the function f. Finally, we require that the rewrite system specified by $\mathcal{F}$ is confluent and terminating [BN98].

The restriction operator $(\nu \overrightarrow{c}; W)P$ declares the names $\overrightarrow{c} = (c_1,\ldots,c_n)$ as fresh in P and states a set of beliefs W on them. A belief $w \in W$ has the form $(c_i,\ldots,c_{i+k}) \in \mathcal{R}$, asserting that the tuple $(c_i,\ldots,c_{i+k})$ is in the relation $\mathcal{R}$. Given a restriction $(\nu \overrightarrow{c}; W)$, we require that $\mathsf{fc}(W) \subseteq \mathsf{fc}(\overrightarrow{c})$. In § 6.2 relations will be introduced that contain symmetric keys (unary) and asymmetric key pairs (binary). In the following, W will denote the set of beliefs stated in a system so far, and we will call it *world*. A regular restriction is obtained specifying no belief on the restricted term, and it will be denoted by $(\nu \overrightarrow{c})$.

The world W plays a key role in evaluating function applications, as the side condition of a rewrite rule can test whether or not some parameters are in a relation $\mathcal{R}$. Such a condition holds if the given relation is in W, as required by the semantics of § 6.3. It is worthwhile noting that a function application may have different evaluations in different worlds.

Processes and quality binders. As for the remaining operators, $e_1!e_2$ represents an asynchronous output of an expression e_2 on channel e_1. The input

$e_1?x[e_2]$ waits for a message on channel e_1 and binds it to variable x if the expression e_2 evaluates to $\mathsf{some}(v)$. When e_2 is a constant other than none we obtain the standard input operator $e?x$; when e_2 contains x we obtain an input operator able to select messages with specific properties: $e?x[\mathsf{fst}(x)]$, for example, accepts only pairs whose first component is not none. This is a very useful feature in a broadcast calculus, in particular when modelling system communicating over a single channel, as we will see in § 6.4. $P_1|P_2$ is the parallel composition of two processes, and $A(e)$ is a call to a process defined in the system, with e being the actual parameter.

Finally, quality binders $\&_q(e_1?x_1[e'_1], \ldots, e_n?x_n[e'_n])$ behave as in the previous versions of the calculus, the main novelty being that the quality guard q is now a place-holder for a function application $f(e_1, \ldots, e_m)$, as explained in § 6.2.

Well-formedness. As for typing, we require that

- in $e_1!e_2$ the channel e_1 and the outputted value e_2 have type V;
- in $e_1?x[e_2]$ the channel e_1 has type V, the input expression (or condition) e_2 has type $V^?$, and the input variable x has type $V^?$;
- in the call $A(e)$ the expression e has type $V^?$;
- in $\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y) : P_1\ \mathsf{else}\ P_2$ the expression e has type $V^?$, and the variable y has type V.

As the syntax is overly liberal in a number of respects, some restrictions help design well-formed processes. First, we will assume that expressions and processes are well-typed and that processes are closed (no free variable). Secondly, we assume that input expressions and quality guards contain only variables that have been defined prior to the binder in which they occur (i.e., in the syntactic prefix). for instance, the process $\&_q(e_1?x_1, e_2?x_2[f(x_1)])$ is not well-formed, as the input on e_2 may arrive before the input on e_1, and in this case we would not be able to evaluate the input condition $f(x_1)$. Finally, limitations apply also to quality guards, as discussed in the following section.

6.2 Exploting Rewrite Rules

6.2.1 Cryptographic reasoning

In the literature about security protocols, since the introduction of the Spi-Calculus [AG97], a popular approach suggests to represent secret cryptographic keys as fresh names. This is for example the case of a symmetric key, or of a private key in an asymmetric cryptosystem. A public key *pub* can then be represented as a function application generated over a free algebra of terms,

and the connection with the private key *prv* is achieved by letting the latter be an argument of the function representing *pub*. This technique, pioneered in the applied π-calculus [AF01], has been used as a successful best practice for encoding security protocols in the verifier ProVerif [Bla09], where cryptographic primitives are implemented as rewrite rules. For example, the rule

$$\mathsf{adec}(\mathsf{aenc}(y_1, pk(y_2)), y_2) \to y_1$$

states that the decryption of a term y_1 encrypted under the public key $pk(y_2)$ is successful only if the third argument is the private key y_2, whose correspondence to $pk(y_2)$ is automatically handled via pattern matching. The main advantage of this approach is that an implementation can directly rely on syntactic unification, that is an efficient procedure. As a main drawback, however, this paradigm fails to express how real cryptosystems work: unless a type system is enforced, any term can be used as key, while in the real world precise constraints apply to keys and to the key generation process. In the following, we overcome this limitation introducing explicit relations between keys, to be exploited in side conditions of rewrite rules. This is achieved without overly complicating the type system of the AQC.

Two simple relations are used to state that a name is a key:

- $c_1 \bowtie c_2$, meaning that $(c_1, c_2) \in \bowtie$ is a pair of keys in an asymmetric cryptosystem; we assume that c_1 is the private key and c_2 is the corresponding public key;
- $c\ltimes$, meaning that c is a key in a symmetric cryptosystem.

For instance, the process $(\nu c_1, c_2; c_1 \bowtie c_2)P$ declares c_1, c_2 as a new key pair in P, and in the trailing process every function application will be evaluated in the world $W = \{c_1 \bowtie c_2\}$. It is worthwhile observing that we could design a process which relies on various cryptosystems simply introducing different relation symbols. On the basis of these relations, a theory for cryptographic primitives is displayed in Table 6.2, in the wake of [Bla02], that pioneered the rewriting approach to the symbolic modelling of cryptographic primitives (in turn inspired by the applied π-calculus). The main novelty of our approach is the use of conditions in rewrites for identifying keys.

Symmetric cryptography. Shared-key encryption can be represented with the symbol $(\mathsf{enc} : V \times V \to V) \in \Sigma$, the first parameter being the message to encrypt and the second one being the key. The corresponding decryption operator is represented by the symbol $(\mathsf{dec} : V \times V \to V^?) \in \Sigma$, which takes an encrypted message as first parameter and a key as second parameter. Observe that a decryption may fail, as dec is allowed to return none. The behaviour of

these operators is modelled by the following rewrite rules in $\mathcal{F}$:

$$\begin{array}{ll} \mathsf{dec}(\mathsf{enc}(y_1, y_2), y_2) \to \mathsf{some}(y_1) & \text{if } y_2 \ltimes \\ \mathsf{dec}(y_1, y_2) \to \mathsf{none} & \text{otherwise} \end{array}$$

These rules state that an encryption is successful only if the key y_2 is in $\ltimes$, i.e., if it is a symmetric key. This approach is coherent with the majority of modern cryptosystems, where keys are required to have particular properties, and thus are distinguishable from random names. In the Advanced Encryption Standard (AES), for example, a valid key must have a predefined dimension (128, 192, or 256 bits). Observe that the second rule applies when at least one of the following condition is met: (i) the first argument is not an encrypted term; (ii) the second argument is not a symmetric key; (iii) the keys used for encryption and decryption do not match. Intuitively, the second rule is applied only if the first rule cannot be applied. The formal treatment of otherwise relies on a transformation that translates a theory containing this keyword into a semantically equivalent theory without this attribute, as explained in [CDE+11, Sec. 4.5.4] . Perfect cryptography is realised if no more rule is provided which deals with encrypted expressions.

Rules can be defined to one's liking. We could be even more restrictive about encryption, requiring $\mathsf{enc}(x, y)$ to be generated with a valid y:

$$\begin{array}{ll} \mathsf{enc}(y_1, y_2) \to \mathsf{some}(\underline{\mathsf{enc}(y_1, y_2)}) & \text{if } y_2 \ltimes \\ \mathsf{enc}(y_1, y_2) \to \mathsf{none} & \text{otherwise} \end{array}$$

and then use $\underline{\mathsf{enc}(y_1, y_2)}$ in the decryption rules (the sort of enc should be changed to allow returning none). In this way we could attain a more precise modelling of a particular cryptosystem.

Asymmetric cryptography. Asymmetric encryption and decryption can be represented with the symbols $(\mathsf{aenc} : V \times V \to V), (\mathsf{adec} : V \times V \to V^?) \in \Sigma$, related by the following rewrite rules in $\mathcal{F}$:

$$\begin{array}{ll} \mathsf{adec}(\mathsf{aenc}(y_1, y_2), y_3) \to \mathsf{some}(y_1) & \text{if } y_3 \bowtie y_2 \\ \mathsf{adec}(y_1, y_2) \to \mathsf{none} & \text{otherwise} \end{array}$$

again, this encoding realises perfect cryptography. The side condition of the first rule requires that the keys y_2 (public) and y_3 (private) come from a valid key pair, i.e., are in the relation defined by $\bowtie$. As for the symmetric case, in fact, asymmetric cryptosystems lay down precise conditions that a key pair has to fulfil.

Signature. A message signed under a secret key is represented with the function $(\mathsf{sign} : V \times V \to V) \in \Sigma$, and we have two corresponding functions for manipulating the generated signed message: $(\mathsf{getmessage} : V \to V^?) \in \Sigma$ returns

Table 6.2: A conditional rewrite theory for cryptographic reasoning.

Σ	$\mathcal{F}$
$\mathsf{enc}: D \times V \to V$ $\mathsf{dec}: V \times V \to V^?$	$\mathsf{dec}(\mathsf{enc}(y_1, y_2), y_2) \to \mathsf{some}(y_1)$ if $y_2 \ltimes$ $\mathsf{dec}(y_1, y_2) \to \mathsf{none}$ otherwise
$\mathsf{aenc}: V \times V \to V$ $\mathsf{adec}: V \times V \to V^?$	$\mathsf{adec}(\mathsf{aenc}(y_1, y_2), y_3) \to \mathsf{some}(y_1)$ if $y_3 \bowtie y_2$ $\mathsf{adec}(y_1, y_2) \to \mathsf{none}$ otherwise
$\mathsf{sign}: V \times V \to V$ $\mathsf{getmessage}: V \to V^?$ $\mathsf{checksign}: V \times V \to V^?$	$\mathsf{getmessage}(\mathsf{sign}(y_1, y_2)) \to \mathsf{some}(y_1)$ $\mathsf{getmessage}(y) \to \mathsf{none}$ otherwise $\mathsf{checksign}(\mathsf{sign}(y_1, y_2), y_3) \to \mathsf{some}(y_1)$ if $y_2 \bowtie y_3$ $\mathsf{checksign}(y_1, y_2) \to \mathsf{none}$ otherwise
$\mathsf{hash}: V \to V$	
$\mathsf{pair}: V \times V \to V$ $\mathsf{fst}: V \to V^?$ $\mathsf{snd}: V \to V^?$	$\mathsf{fst}(\mathsf{pair}(y_1, y_2)) \to \mathsf{some}(y_1)$ $\mathsf{snd}(\mathsf{pair}(y_1, y_2)) \to \mathsf{some}(y_2)$ $\mathsf{fst}(y) \to \mathsf{none}$ otherwise $\mathsf{snd}(y) \to \mathsf{none}$ otherwise

the message without verifying the signature, while $(\mathsf{checksign}: V \times V \to V^?) \in \Sigma$ performs the check. The rewrite rules follow which relate these symbols:

$$
\begin{array}{ll}
\mathsf{getmessage}(\mathsf{sign}(y_1, y_2)) \to \mathsf{some}(y_1) & \\
\mathsf{getmessage}(y) \to \mathsf{none} & \text{otherwise} \\
\mathsf{checksign}(\mathsf{sign}(y_1, y_2), y_3) \to \mathsf{some}(y_1) & \text{if } y_2 \bowtie y_3 \\
\mathsf{checksign}(y_1, y_2) \to \mathsf{none} & \text{otherwise}
\end{array}
$$

observe that we will always be able to retrieve the original message when we do not verify the signature, thus the rule for getmessage has no side condition and behaves identically in every world. In contrast to this, checksign requires that (y_2, y_3) is a a valid private-public key pair.

It is worth noting that we are assuming $\bowtie$ to exhibit a particular property: the private key can be used for encryption, as in RSA. We could model asymmetric cryptosystems which do not have this property, like the Digital Signature Algorithm (DSA), introducing a different symbol for relating keys.

Hash. An hashing function can be represented with the symbol $(\mathsf{hash}: V \to V) \in \Sigma$, without related rule in $\mathcal{F}$, for an hashed message cannot be inverted in our ideal world.

Table 6.2 summarises the rules presented so far and gives pairing and projections constructs that we shall use in the example of § 6.4.

6.2.2 Quality guards

Quality guards decide when a quality binder is satisfied and the trailing process can be executed. Let $\mathcal{B}$ be a sub-type of V representing Booleans, where the truth values $\{\mathsf{ff}, \mathsf{tt}\}$ are constants defined in the signature Σ. A quality guard $q(e_1, \ldots, e_m)$ for a binder $\&_{q(e_1,\ldots,e_m)}(b_1, \ldots, b_n)$ is a function $(q{:}t_1 \times \cdots \times t_m \times \mathcal{B}^n \to \mathcal{B}) \in \Sigma$ which takes as parameters

- m expressions $e_1, \ldots, e_m$ with types $t_1, \ldots, t_m$ (either V or $V^?$),
- n Boolean parameters, each one stating whether or not a b_i has been satisfied,

and returns tt, in which case the binder is satisfied and its continuation is evaluated, or ff otherwise. A simple input evaluates to tt if it is performed, i.e., it transforms into a substitution, while it gives ff if at the time of the evaluation it has not been performed yet. As the n parameters related to the sub-binders are always present, we omit them for the sake of brevity.

In the Quality Calculus guards involve only the status of the sub-binders and are specified with predicates, denoted by $\forall$, $\exists$, $\exists!$, m/n, requiring to perform all the inputs, one input, exactly one input, or m out of n possible inputs before passing a binder, respectively. Rewrite rules can express predicates but allow to design also more elaborate guards. We can legally write, for example,

$$P \triangleq \&_{\exists}(c_1?x_1, c_2?x_2).\&_{q(x_1,x_2)}(c_1?x_3, c_2?x_4)$$

where the guard q is defined by the rule

$$q(x_1', x_2', y_1', y_2') \to (\mathsf{issome}(x_1') \wedge y_1') \vee (\mathsf{issome}(x_2') \wedge y_2')$$

The first two formal parameters of q are the input variables on which the first quality binder in P ranges, while the latter correspond to the Boolean interpretation of the inputs of the second quality binder in P. The quality guard q states that the condition for consuming the second quality binder depends on the outcome of the first quality binder: if only the input concerning x_1 was performed, then x_3 must be bound to $\mathsf{some}(c)$, that is, y_1 must be tt, and conversely x_4 must be $\mathsf{some}(c)$ if only the input involving x_2 was received. In this sense, we may speak of q as a *relational* quality guard, which allows to write even more compact specifications.

The use of function $\mathsf{issome}(\cdot)$, checking whether its argument is none, is crucial, as we can ask whether or not an input has been performed, but we should not inspect its content in a quality guard. For the sake of discussion, assume that a guard decides whether or not to pass a quality binder inspecting the content of received inputs: in this case we might end up in a situation in which all the inputs have been performed but the binder cannot be consumed due to what we received, and the process would be stuck since we have no means

to re-bind an input. Input expressions are entrusted of selecting inputs on the basis of their content, and such a test should be carefully avoided in quality guards. As a matter of fact, we can model the desired behaviour in a safe way relying on input conditions, rewriting the second quality binder of P as follows:

$$\&_{q(x_1,x_2)}(c_1?x_3[x_1 = x_3)], c_2?x_4)$$

where the equality between the two inputs on c_1 is enforced by the input expression $x_1 = x_3$, formally defined in the following section.

6.3 Semantics

The semantics of the AQC is defined by a structural equivalence $\equiv$ and a labelled transition relation $\Longrightarrow$. The standard structural congruence is adapted to the new restriction operator, hence condition on restricted names are replaced by condition on list of names. Likewise, in the syntax of contexts C the format $(\nu\overrightarrow{c}; W)C$ replaces $(\nu c)\, C$. As usual, we assume to apply α-conversion whenever needed, in order to avoid accidental captures.

Auxiliary relations. The semantics is parametrised on some auxiliary relations. The first two, defined formally in Table 6.5, are the standard relations for I/O synchronisation and binder evaluation common to Quality Calculi:

- $W \vdash e_1!e_2|b \to b'$, specifying the effect of the output $e_1!e_2$ on the binder b in the world W; technically, the syntax of binders has to be extended to consider substitutions of the form $[e/x]$;
- $b ::_r \theta$, for recording in $r \in \{\mathsf{ff}, \mathsf{tt}\}$ whether or not the binder b has been satisfied by the received inputs, that led to the substitutions recorded in θ. Observe that r has type $\mathcal{B}$, and thus can be passed as argument to a rewrite rule specifying a quality guard (Boolean interpretation of binders);
- $W \vdash e \rhd e'$, describing how an expression e evaluates to a constant expression e' (either a v or an o) in the world W, according to the rules displayed in Table 6.3

Finally, Table 6.3 shows how expressions are evaluated in a given world. A name c and the constant none always evaluate to themselves in every world. A world W supports a rewrite step applied to the actual parameters $e'_1, \ldots, e'_n$ if the side condition holds in W. A function application $f(e'_1, \ldots, e'_n)$ is evaluated applying rewriting steps until a non-reducible expression u is produced (i.e., a normal form). Observe that a rewrite step is viable only if there exists a rule $f(e_1, \ldots, e_n)$ if *cond* in $\mathcal{F}$ such that (*i*) the leftmost symbol corresponds to f (and arity and sort match), (*ii*) the formal parameters unify with the actual parameters under a most general unifier θ, (*iii*) the side condition (if any) holds

Table 6.3: The evaluation $W \vdash e \rhd o$ of expressions into optional values.

$$\vdash c \rhd c \qquad \vdash \mathsf{none} \rhd \mathsf{none}$$

$$\frac{(f(e_1,\ldots,e_n) \to e \text{ if } cond) \in \mathcal{F} \qquad W \vdash \theta(cond)}{W \vdash f(\theta(e'_1),\ldots,\theta(e'_n)) \to \theta(e)}$$

$$\frac{W \vdash f(e_1,\ldots,e_n) \to^* u \quad \nexists u'.W \vdash u \to u'}{W \vdash f(e_1,\ldots,e_n) \rhd u}$$

in W after having undergone the substitutions in θ. If one of these conditions does not hold, then the evaluation fails and the process is stuck (the condition otherwise helps design robust function specification). $\to^*$ denotes the reflexive transitive closure of the rewriting relation $\to$, and $W \vdash t \to^* t'$ if W supports all the side conditions of at least one rewrite path that reduces t to t'. Since we require the rewrite system to be confluent and terminating, the evaluation strategy supporting the computation does not affect the result – that is, normal forms are unique –, even if it can heavily impact the performance.

We can now define $\mathsf{issome}\colon V^? \to \mathcal{B}$ formally :

$$\mathsf{issome}(x) \to \mathsf{tt} \text{ if } x \rhd \mathsf{some}(v) \qquad \mathsf{issome}(x) \to \mathsf{ff} \text{ if } x \rhd \mathsf{none}$$

The expression evaluation relation $\rhd$ allows also to define equality modulo rewriting, represented by the symbol $(=\colon V^{?^2} \to V^?) \in \Sigma$:

$$\begin{array}{ll} =(x_1,x_2) \to \mathsf{some}(\mathsf{tt}) & \text{if } x_1 \rhd o \wedge x_2 \rhd o \\ =(x_1,x_2) \to \mathsf{none} & \text{otherwise} \end{array}$$

In the following we will use the infix notation for the sake of simplicity.

Transition relation. The transition relation $P \Longrightarrow P'$ describes when a process P evolves into another process P'. $P \Longrightarrow P'$ is based on the relation $W \vdash P \xrightarrow[i]{\alpha} P'$, which is instrumented with a label α and an integer i. The label α is either the silent action τ or an output action $c_1!v_2$, while the integer i is either 0 (passive action) or 1 (active action). The relations are defined according to rules displayed in Table 6.4. An active step $\xrightarrow[1]{\alpha}$ enables a transition $\Longrightarrow$, as explained by rule (Sys), which also takes care of pulling restrictions to the outer-most level by means of the structural congruence. The set W contains the beliefs accumulated with restrictions, and the second rule allows to evaluate rewrite conditions. As the world W affects expression and binder evaluation, it is a key component in determining the path followed by the computation: the

possibility to execute a step $\xrightarrow[i]{\alpha}$ may depend on W. In order to stress this relationship, we write $W \vdash P \xrightarrow[i]{\alpha} P'$.

The second group of clauses defines $\xrightarrow[i]{\alpha}$. Rule (Self) states that a process can silently evolve to itself, and it is central in enabling non-communicating processes to interleave. It is worthwhile observing that such a rule corresponds to an unguarded recursion and therefore results in divergent behaviour [Hoa85, § 3.8]. Nonetheless, as the action is passive, such a step never gives rise to a global transition $\Longrightarrow$, hence divergence is filtered out by the semantics. Rule (Brd) describes how performing an output is possibly an asynchronous active action (and thus can directly turn into a step $\Longrightarrow$). As in Ch 5, we require the output channel to be a name c and we let the broadcast message by a value v. The following two rules describe how an output affects a process guarded by a binder: if the output does not satisfy the binder (In-ff), then the related substitution is recorded and the binder modified accordingly; otherwise, if the output satisfies the binder (In-tt), the substitution computed so far is applied to the continuation process and the binder consumed. Observe that in both cases the action is passive, and therefore cannot directly enable a $\Longrightarrow$ transition. This happens only when a synchronisation step takes place, as described by rule (Par). The composition of two processes that can make a transition $\xrightarrow[i]{\alpha}$ evolves only if the processes share the label α and the transitions are not both active. This implies that two input processes waiting for a value on the same channel can evolve together, as well as two processes that can synchronise: the synchronisation with an output process is actually the only case in which an input transition transforms into an active action, giving rise to a transition $\Longrightarrow$. Thanks to this behaviour, the semantics realises broadcast communication: a binder accumulates expected outputs and the related substitutions, and when the synchronisation finally takes place with a matching output all the involved binders evolve at the same time. Observe that two outputs cannot evolve simultaneously, as they both are active actions. Rules (Case-) describe the evaluation of a case construct: the else branch is taken whenever the expression evaluates to none, otherwise the computation proceeds binding to the variable y the value to which the expression evaluates. Observe that a case transition is always labelled with τ, and thus cannot be mixed with input or output actions when composing two processes. Finally, rule (Rec) accommodates recursive calls.

In particular, observe that no rule is given to perform transitions under a restriction, therefore restrictions must be pulled at the outer-most level by means of structural rules, as dictated by rule (Sys). Again, we focus on closed systems, the semantic being defined by $\Longrightarrow$.

Table 6.5 presents the usual auxiliary relations for synchronisation and binder evaluation. The clauses in the first group present how I/O substitutions are computed. An output affects an input binder only if they are performed on the same channel (which must be a name) and the input condition evaluates to

Table 6.4: The transition relation $\Longrightarrow$ of the Applied Quality Calculus.

$$\frac{P_1 \equiv (\nu \overrightarrow{a}; W)P_2 \quad W \vdash P_2 \xrightarrow[1]{\alpha} P_3 \quad (\nu \overrightarrow{a}; W)P_3 \equiv P_4}{P_1 \Longrightarrow P_4}\ (\text{Sys}) \qquad \frac{w \in W}{W \vdash w}$$

$$\vdash P \xrightarrow[0]{\tau} P\ (\text{Self}) \qquad \frac{W \vdash e_1 \rhd c_1 \quad W \vdash e_2 \rhd v_2}{W \vdash e_1!e_2.P \xrightarrow[1]{c_1!v_2} P}\ (\text{Brd})$$

$$\frac{W \vdash c_1!v_2 | b \to b' \quad b' ::_{\mathsf{ff}} \theta}{W \vdash b.P \xrightarrow[0]{c_1!v_2} b'.P}\ (\text{In-ff})$$

$$\frac{W \vdash c_1!v_2 | b \to b' \quad b' ::_{\mathsf{tt}} \theta}{W \vdash b.P \xrightarrow[0]{c_1!v_2} P\theta}\ (\text{In-tt})$$

$$\frac{W \vdash P_1 \xrightarrow[i]{\alpha} P_1' \quad W \vdash P_2 \xrightarrow[j]{\alpha} P_2'}{W \vdash P_1 \mid P_2 \xrightarrow[i+j]{\alpha} P_1' \mid P_2'} \quad i + j \leq 1\ (\text{Par})$$

$$\frac{W \vdash e \rhd \mathsf{some}(v)}{W \vdash \mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y) \colon P_1\ \mathsf{else}\ P_2 \xrightarrow[1]{\tau} P_1[v/y]}\ (\text{Case-tt})$$

$$\frac{W \vdash e \rhd \mathsf{none}}{W \vdash \mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y) \colon P_1\ \mathsf{else}\ P_2 \xrightarrow[1]{\tau} P_2}\ (\text{Case-ff})$$

$$\frac{}{A(e) \xrightarrow[1]{\tau} P[e/x]} \quad \text{if } A(x) \triangleq P\ (\text{Rec})$$

values when the outputted value is substituted to the input variable; otherwise it has no effect.[1] Observe how broadcasting is realised within a single quality binder: sub-binders can be affected by the same output. This behaviour is coherent with the intended semantics of the quality binder, which states that n inputs are *simultaneously active* in $\&_q(e_1?x_1[e'_1], \ldots, e_n?x_n[e'_n])$, and therefore a number of them can synchronise with a single matching output.

The second group of clauses shows how a binder is evaluated, complementing the input clauses of the second group. In particular, a substitution evaluates to tt, since it is the result of a successful input, while a non-performed input evaluates to ff and maps the input variable to none. A quality binder is evaluated computing the function specified by the quality guard.

Finally, we require that a process is always in a stable configuration, according to which all the internal (silent) actions are performed before the possibility to synchronise with an external output vanishes. This implies that the process

$$c_1!c_2 \mid \mathsf{case}\ \mathsf{some}(c_1)\ \mathsf{of}\ \mathsf{some}(y) : y?x.P\ \mathsf{else}\ Q$$

will always evolve to $P[\mathsf{some}(c_2)/x]$. Intuitively, by imposing this restriction we assume that internal actions are always processed faster than communicating actions (an on-board processor is faster than a transceiver).

6.4 Key Update through Secret Sharing

We demonstrate the flexibility of the AQC modelling a hierarchical WSN where secret sharing is exploited to communicate security-critical information to a base station, as studied in [SLK10].

In such a scenario, each message is broadcast over a single wireless channel and can thus be eavesdropped by an attacker. A (k, m)-threshold-scheme can be applied to security-critical messages in order to hamper the work of the adversary, who is required to intercept at least k shares (or shadows) before obtaining a message (and trying to break an encryption scheme). Moreover, we assume that the communication may fail, due to environmental conditions, hardware failures, or the attacker's intervention, and we exploit quality binders in order to design processes robust against DoS. Finally, the base station is periodically receiving data from sensor nodes in its range, which measure some physical parameters of the environment. Nonetheless, when updating the session key, the base station ignores messages sent by the sensors, in order to accomplish this critical task as quickly as possible. As all the communications take place on a single channel, we make use of input conditions to distinguish among messages.

[1] Note that only for the purpose of evaluating the input condition we temporarily bind the input variable x to the value v_2: all the input conditions we devise in this chapter takes values as input and therefore demand for this choice. The alternative choice of optional values would work likewise. Observe that if the matching is not successful than such binding has no effect on the continuation.

Table 6.5: The relations for synchronisation and binder evaluation.

$$\frac{W \vdash e_1 \rhd c_1 \quad W \vdash e_2[v_2/x] \rhd \mathsf{some}(c_3)}{W \vdash c_1!v_2 | e_1?x[e_2] \to [\mathsf{some}(v_2)/x]}$$

$$\frac{W \vdash e_1 \rhd c_1 \quad W \vdash e_2[v_2/x] \rhd \mathsf{none}}{W \vdash c_1!v_2 | e_1?x[e_2] \to e_1?x[e_2]}$$

$$\frac{W \vdash e_1 \rhd c_1'}{W \vdash c_1!v_2 | e_1?x[e_2] \to e_1?x[e_2]} \quad c_1' \neq c_1$$

$$\frac{W \vdash c_1!v_2 | b_1 \to b_1' \cdots W \vdash c_1!v_2 | b_n \to b_n'}{W \vdash c_1!v_2 | \&_q(b_1, \ldots, b_n) \to \&_q(b_1', \ldots, b_n')}$$

$$e_1?x[e_2] ::_{\mathsf{ff}} [\mathsf{none}/x] \qquad [\mathsf{some}(v)/x] ::_{\mathsf{tt}} [\mathsf{some}(v)/x]$$

$$\frac{b_1 ::_{r_1} \theta_1 \quad \cdots \quad b_n ::_{r_n} \theta_n}{\&_{q(e_1,\ldots,e_m)}(b_1, \ldots, b_n) ::_r \theta_n \cdots \theta_1} \quad q(e_1, \ldots, e_m, r_1, \ldots, r_n) \rhd r$$

Secret sharing. A shadow is represented by function $(\mathsf{share} : V^3 \to V) \in \Sigma$, the first parameter being the secret, the second the number of shares needed for reconstructing it, and the third an identifier that helps distinguish different shares. The reconstruction step is modelled with function $(\mathsf{combine} : V^k \to V^?) \in \Sigma$, which takes k shares and returns the secret only if they are all different. For fixed $k = m = 3$, we obtain the following implementation:

$$\mathsf{combine}(\mathsf{share}(y_1, 3, y_1'), \mathsf{share}(y_1, 3, y_2'), \mathsf{share}(y_1, 3, y_3')) \to \mathsf{some}(y_1) \quad \text{if } y_i' \rhd v_i \wedge i \neq j \Rightarrow v_i \neq v_j$$

$$\mathsf{combine}(y_1, y_2, y_3) \to \mathsf{none} \quad \text{otherwise}$$

We omit the parameter m for the sake of succinctness, but it could be included to capture the fact that shadows of a same secret s belonging to different schemes cannot be used to rebuild s.

The protocol. Consider now a WSN in which a central unit CU has to communicate a new symmetric session key to a base station under its control. CU generates a new symmetric key k, signs it with its secret key sk_{CU}, computes 3 shares, encrypts the shadows under the base station public key pk_{BS}, and finally communicates the shares on a wireless channel c after having notified the base

station that an update transaction is starting:

$$
\begin{aligned}
\mathsf{CU} \triangleq\ & (\nu k; k \ltimes)\mathsf{c!start\_transaction} \\
& \mathsf{c!pair}(\mathsf{aenc}(\mathsf{share}(\mathsf{sign}(k, sk_{\mathsf{CU}}), 3, 1), pk_{\mathsf{BS}}), 1) \\
& \mathsf{c!pair}(\mathsf{aenc}(\mathsf{share}(\mathsf{sign}(k, sk_{\mathsf{CU}}), 3, 2), pk_{\mathsf{BS}}), 2) \\
& \mathsf{c!pair}(\mathsf{aenc}(\mathsf{share}(\mathsf{sign}(k, sk_{\mathsf{CU}}), 3, 3), pk_{\mathsf{BS}}), 3) \\
& \mathsf{set}_1!\mathsf{t}_1.\&_{\exists}(\mathsf{c}?x_e[\mathsf{dec}(x_e, k) = \mathsf{some}(\mathsf{end\_transaction})], \mathsf{tick}_1?x_{t1}) \\
& \mathsf{case}\ x_e\ \mathsf{of}\ \mathsf{some}(y_e) : \mathsf{set}_2!\mathsf{t}_2.\mathsf{tick}_2?x_{t2}.\mathsf{CU}\ \mathsf{else}\ \mathsf{CU}
\end{aligned}
$$

The process CU makes use of numbers (constants) to represent integers (threshold-scheme parameters and share identifiers). After having issued the new key (signed and then split in three encrypted shadows), the central unit sets a local timer to t_1 and waits for a notification from the base station to arrive within the prescribed time. The time expires when a message is received from the timer on channel tick_1. Observe that the first input is instrumented with a condition that tests whether or not the received message corresponds to end_transaction encrypted under the new key k. At this point the central unit is enabled to check which input triggered passing the binder, and thus decide if the key update was successful or not: in the event the key has been updated, the central unit waits for t_2 unit of time and then starts a new update transaction, otherwise the new transaction is started immediately. A new timer is used to avoid message confusion. Observe that there is no need to decrypt x_e since its content has already been tested in the input condition: we only need to check that it is not none.

It is worth noting that the central unit is robust with respect to the event that the base station does not respond, thanks to the use of the existential quality guard and of the local clock, which always responds.

The base station is defined by a process BS, which waits for three shares and then sends an acknowledgement back to CU, encrypted under the new key.

$$
\begin{aligned}
\mathsf{BS} \triangleq\ & \mathsf{c}?x.\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(y_1) : \\
& \quad \mathsf{case}\ y_1 = \mathsf{end\_transaction}\ \mathsf{of}\ \mathsf{some}(y_2) : \mathsf{set}_3!\mathsf{t}_3 \\
& \quad\quad \&_{\exists}(\&_{\forall}(\mathsf{c}?z_1[\mathsf{snd}(z_1) = \mathsf{some}(1)], \mathsf{c}?z_2[\mathsf{snd}(z_2) = \mathsf{some}(2)] \\
& \quad\quad\quad\quad \mathsf{c}?z_3[\mathsf{snd}(z_3) = \mathsf{some}(3)]), \\
& \quad\quad\quad\quad \mathsf{tick}_3?x_{t3}) \ldots \textit{(extract the shares in } y'_1, y'_2, y'_3\textit{)}\ \ldots \\
& \quad\quad\quad\quad\quad \mathsf{case}\ \mathsf{combine}(y'_1, y'_2, y'_3)\ \mathsf{of}\ \mathsf{some}(y_s) : \\
& \quad\quad\quad\quad\quad\quad \mathsf{case}\ \mathsf{checksign}(y_s, pk_{\mathsf{CU}})\ \mathsf{of}\ \mathsf{some}(y_k) : \\
& \quad\quad\quad\quad\quad\quad\quad \mathsf{c!enc}(\mathsf{end\_transaction}, y_k).\mathsf{BS}\ \mathsf{else}\ 0 \\
& \quad\quad\quad\quad\quad\quad \mathsf{else}\ 0 \\
& \quad\quad\quad\quad\quad \mathsf{else}\ 0 \ldots \textit{(if share extraction fails then shut down)}\ \ldots \\
& \quad\quad \mathsf{else}\ \mathsf{store}!y_1.\mathsf{BS} \\
& \quad \mathsf{else}\ 0
\end{aligned}
$$

If BS receives a reading from a sensor, the value is stored in the base station memory (simulated by channel store). If an update instruction is received, then the base station waits for three shares. BS must wait for all the shadows to

arrive, and thus uses a quality binder instrumented with the $\forall$ guard. Furthermore, in order to discard messages from the sensors, the inputs within the binder rely on a condition matching only a fixed pattern. For the sake of brevity, we have omitted a number of case constructs needed to compute projections and to decrypt the result. When three messages are received within time t_3, the base station tries to compute the original secret: if the operation succeeds then the signature is verified using the public key pk_{CU} of CU, and then an acknowledgement is sent back to the central unit.

Observe that the base station continues to behave in a planned manner even if the information expected from the central unit does not arrive. If the shares are not received within time t_3, their combination or the verification of the signature fail, then BS is automatically switched off for security reasons.

6.5 Concluding Remarks

The characteristics of typical components of CPSs and the nature of the environment in which they are deployed demand for designing software that is both robust against lacking communication and able to ignore unwanted information. This is a complex task *per se*, and it is even harder in those applications that require some degree of security and need a broadcast communication model, where everyone hears everyone.

The framework we have presented facilitates the design of CPSs by providing a calculus that is naturally equipped with the notion of absence of communication and selective inputs. Denial-of-service is addressed by resorting to the distinction between values and optional values introduced by the Quality Calculus. A single mechanism based on rewrite rules is leveraged to implement both selective inputs and cryptographic reasoning, and it is also exploited to design elaborate quality guards, more expressive than propositional predicates. Moreover, a simple yet powerful approach to the definition of cryptographic material has been introduced. The expressiveness of the framework has been discussed on a meaningful example, in which all its features have been stressed.

Future work includes further investigation of verification techniques based on term rewriting. It seems promising to study abstraction approaches for analysing infinite state systems beyond bounded reachability, in the wake of studies carried out in the Maude community. Moreover, it would be interesting to consider a wider class of CPSs with component mobility, thus enriching the framework with a notion of network topology and spatially-bounded broadcast.

Equational reasoning. Various approaches have been studied in the literature in order to enrich a process calculus by means of equational reasoning, and a relevant line of work is devoted to equip process calculi with cryptographic primitives.

In the Spi Calculus [AG97] cryptographic functions are represented as function applications on terms, while the corresponding inverse functions are modelled with dedicated clauses in the syntax of processes. Even though this approach is clear and simple, it lacks of flexibility: the syntax of terms and processes have to be modified every time a new function or inverse is added to the calculus. This main drawback has been overcome in the applied π-calculus [AF01], where a general notion of composite terms (function applications) is presented, together with a general mechanism for term manipulation. In particular, an algebraic signature is used to generate composite terms. The inverse relation is then obtained instrumenting the signature with an equivalence relation over terms (i.e., an *equational theory*). A single conditional clause $\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$ in the grammar of processes is thus evaluated according to the equational theory: if $t_1 = t_2$ in the theory, then P is executed, otherwise the process evolves to Q. Note that the $=$ symbol represents equality modulo the theory rather than syntactical identity. A further generalisation of the applied-π calculus is given in the Psi-calculi framework [BHJ+11]. Obviously, the flexibility issue which affects the Spi Calculus is partly lifted from the grammar of processes to the equational theory, but the latter can be maintained as an external component: different versions of the calculus are obtained modifying the signature and the equational theory, while syntax and semantics remain unchanged. Moreover, once a verification procedure is devised for such a calculus, it will generally work independently from the signature and the theory.

The process calculus introduced with ProVerif [Bla09] stems directly from the applied π-calculus, from which it differs mainly in the approach to the underlying equational theory, in which equations are replaced by rewrite rules from terms to terms, embedded into processes by a single clause, called *destructor application*. This technique allows to express the theory in a more compact way: there is no assumption about the closure of the set of equations, and equality is checked computing a substitution that matches the actual parameters of a destructor with the formal parameters of the corresponding rewrite rule. On the one hand, the choice of rewrite rules leads to a very efficient verification procedure, since the evaluation of destructors relies on standard unification [BAF08]; on the other hand, the categorisation of functions into constructors (function applications) and destructors exhibits some limitations that equational theories do not: there exist equational theories which cannot be described in this model, e.g., theories containing associative symbols like the exclusive or (XOR). This limitations can be avoided shifting from syntactic unification to unification modulo theories (paramodulation), with an increase in complexity [AB05, Sec. 8].

LySA^{NS} [BRN04] exploits a different technique for building and manipulating complex terms, based on pattern matching. A composite term (function application) can be matched against a *pattern*, which can be used in a process to test syntactical identity and bind sub-terms to variables, extracting values of interest. Syntactically, each composite term and the related patterns must be

introduced explicitly, like in the Spi Calculus. The reader had a sample of the approach in Ch. 5.

Our conditional rewrite rules are inspired by Maude [CDE+11], which unlike ProVerif is a pure rewriting logic tool. For an extensive bibliography about rewriting logic refer to [MOPV12]. In particular, [MOPV12, § 2] lists some works that pioneered the approach we undertake in the paper on which this chapter is based, i.e., complementing the modelling language with an executable specification of the operational semantics by means of an implementation in Maude. Among these, [VMO02, TSMO04, VMO06] are relevant to process calculi. Finally, it is worth mentioning Maude-NPA [EMM09], an extension of Maude targeted to the analysis of security protocols, where backward narrowing is exploited to search for possible attacks in the Dolev-Yao model.

Finally, as for the way cryptographic keys are treated, alternatives to pure restriction have been explored [CGG05, GlPT07]. In the *secret* π-calculus, for example, an ad hoc operator is introduced which declares secrets with static scope. In Ch. 7 we shall briefly relate some of our developments to this line of research.

Broadcast calculi. The Calculus of Broadcasting Systems (CBS) [Pra95] is the ancestor of a number of modern broadcasting calculi. In the subsequent studies, two main strands have flourished: on the one hand, theories for node mobility and dynamic topologies have been investigated; on the other hand, a number of calculi have been proposed that deal with low level characteristics of broadcast communication, such as transmission interference and range.

The calculus presented in [NH06] extends CBS with the notion of topology and presents an analysis for checking its consistency, but does not discuss the representation of cryptographic primitives in detail. Mobility in ad hoc networks is considered in [Mer09], where a labelled characterisation of reduction barbed congruence is proposed. In [GFM10], the dynamics of the topology is implicitly modelled in the semantics of the calculus. Different mobility models are studied in [NG09], together with their relationship to real applications.

The Calculus of Wireless Systems (CWS) [LS10] gives a lower-level representation of communication, modelling transmission interference in the semantics. CWS is extended in [MBS11], where a timed scenario is considered which allows to study communication collisions and CSMA protocols. Interference is also considered in [BGM+12], where a notion of interference-sensitive preorder is introduced for mobile ad hoc networks.

Node mobility is studied together with limited transmission range in [SRS10], combining the two lines of research mentioned above. A similar perspective is adopted in [SG10], which proposes a calculus where node mobility impacts on the reliability of transmissions in a probabilistic fashion. Finally, the calculus of [KP11] investigates different abstraction levels for describing dynamic networks, and considers the possibility of broadcasting at multiple transmission

ranges.

Except for [NH06], none of the calculi mentioned so far provide any kind of equational reasoning on the messages that communicating parties exchange. In [God07] a calculus for mobile ad hoc networks (CMAN) is devised, encompassing node mobility, spatially-oriented broadcast, and an implementation of cryptographic primitives via equations à la applied π-calculus. Recently, [MM12] has proposed a timed process calculus with fixed transmission ranges, where equational reasoning is parametrised on an inference system (our approach based on rewrite theories is similar, but allows a more expressive treatment of cryptographic primitives). Moreover, the calculus is equipped with a simulation theory, and the authors envision a possible mechanisation via Isabelle/HOL or Coq.

As for the analysis of reachability properties in wireless settings, [BHJ+11] presents a broadcast version of the psi-calculi framework with an application to a routing protocol for mobile ad hoc networks.

Chapter 7

Quantifying Protection

So far we have explored how availability can be construed in terms of reachability in a qualitative mind-set. We shall now move towards a quantitative framework, so as to bridge the gap with existing methods for countering DoS, mainly based on cost considerations (cf. § 3.1.3). Remaining in the realm of the Quality Calculus, we compute the cost of reaching a given program point for an attacker by means of a *protection analysis* that can be seen as a quantitative counter-part of the robustness analysis of Ch. 4.

Even though the urgency of this investigation stems from our interest in quantifying the resilience of a system to unavailability, still the developments lead to tackle naturally the ampler problem of guaranteeing multi-level security assurances, and we shall present them in this light for the sake of generality.

In order to compute the cost of reaching a target location l in a given system, we need to first *discover* all paths leading to l, i.e., all attacks, and then to *quantify* the cost of following such paths. The Quality Calculus offers an elegant framework for reasoning about systems where the same functionality can be triggered in multiple ways, and thus enjoy a branching control flow. Taking a process-algebraic point of view, moreover, is suitable for describing software systems but also organisations or physical infrastructure in a uniform manner.

In the study of security, the compromise for obtaining such broad domain coverage while retaining a reasonable expressive power takes place at the level of attack definition. At a high level of abstraction, an attack can be defined as a sequence of actions undertaken by an adversary in order to make unauthorised use of a target asset. In a process-algebraic world, this necessary *interaction* between the adversary and the target system is construed in terms of com-

municating processes. Input actions on the system side can be thought of as *security checks*, that require some information to be fulfilled. In particular, the capability of communicating over a given channel requires the knowledge of the channel itself and of the communication standard. This could include, for example, the knowledge of some cryptographic keys used to secure the communication. Hence, we can think of a compound attack as a set of channels needed to activate the desired behaviour on the target system side, that is, to reach a particular location that should be secured from unauthorised access.

Whilst the correctness of such *secure channels* [MV09] is investigated in the realm of protocol verification, any system with a substantial need for security is likely to have standardised mechanisms to achieve various degrees of protection, and modelling them with secure channels is a coarse yet reasonable abstraction.

Given a system P and a location l of interest, the protection analysis computes the information an adversary needs in order to drive P to l, considering all possible paths leading to l. This is achieved by translating P into a set of propositional formulae describing the dependency between channels and reachability of locations: each satisfying assignment of such a set of formulae describes a way in which l can be attained in terms of communication over channels (§ 7.3). In order to match the abstraction to secure channels, the analysis is developed on a value-passing version of the calculus (§ 7.1).

Nonetheless, secure channels allow modelling a great many domains. In IT systems, a channel can be thought of as a wired or wireless communication link over which messages are encrypted, and its knowledge mimics the knowledge of a suitable encryption key. In the physical world, a channel can represent a door, and its knowledge the ownership of the key, a pin code, or the capability of bypassing a retinal scan. This urges to acknowledge the variety of protection mechanisms and devise *attack metrics* by assigning costs to channels, thus quantifying the effort required to an attacker for obtaining the channel or, equivalently, the protection guarantees ensured by the security mechanism represented by the channel. As a consequence, attacks can be ordered according to their cost. Finally, a conservative approach to security demands to look for attacks that are *optimal* in the cost ordering (that is, minimal or maximal, according to the notion of cost we adopt – in the following we shall focus without loss of generality on minimality).

The protection analysis is then lifted to a quantitative setting (§ 7.4), where the quest for minimal models (i.e., the cheapest attacks) is implemented on top of an SMT solver, where minimality is sought with respect to an objective function defined on a liberal cost structure in which both symbolic and non-linearly-ordered cost sets can be represented.

The ultimate goal of the framework is to check whether the protection ensured by the implementation, that is, the cheapest way for reaching l, matches the specification requirements, formalised as a map from locations to desired confidentiality levels. Narrowing the picture down to DoS, the protection analysis would allow to estimate the cost an attacker incurs to drive a system to

an unavailability condition, e.g., the terminated process 0 of the base station formalised in § 4.4.

The novelty of the protection analysis is many-fold. First of all, the problem of inferring attacks and quantifying the protection offered by security checks is interesting *per se* and poorly addressed in the literature. In particular, we devise a comprehensive approach, from modelling systems and their security architecture to mechanising the verification of how this architecture has been realised in the implementation. Moreover, the extension from qualitative to quantitative settings can be mimicked in a great many contexts, and it is seamlessly applicable to other formal specifications that resort to encoding into logic formulae. In connection with this, our SMT-based solution procedure can be applied to all problems requiring to rank models of a logic formula according to given criteria.

The quest for optimal attacks can cope with arbitrary cost structures and objective functions, whose shape is only limited by the expressiveness of modern SMT solvers. Whilst our SMT approach to optimisation is not entirely novel, non-linearly-ordered and symbolic cost structures have not been addressed so far in connection with this technique.

Finally, the analysis set bridges between the qualitative approach to DoS developed so far and the quantitative mind-set of Meadows's unifying framework (cf. 3.2.2). In particular, observe that in Meadows's work the target threshold relation defining potential DoS consists of pairs of costs to the defender and to the attacker of a system with respect to a given program point (in Meadows's formalisation, the end of a protocol). Whilst in Meadows's work the attacker is fully-specified, being a legal participant to the protocol, we only require to specify the defender (i.e., the system of interest), and automatically synthesise a set of "cheapest attackers" as the *co-processes* that can drive the system to a given point (also known as *catalysers*). On the contrary, we do not compute the cost of operating the system on the defender side, assuming that this ought to be a simpler job the system being under control of the designer. However, it seems to us that an identical technique could be exploited to come up with an estimation of the cost to the defender.

The usefulness of the framework is demonstrated on the study of password recovery systems, formulated in § 7.2 and then studied throughout the chapter. We consider the *Microsoft account* system, where a challenge is sent to a previously registered e-mail or phone number, and the *Yahoo!Mail* system, which allows resetting a password upon answering previously selected questions. The results we present are obtained through an implementation of the analysis freely available.

This chapter is based on [VNR14c] and on material currently under submission.

Table 7.1: The syntax of the Value-Passing Quality Calculus.

$$
\begin{array}{rcl}
P & ::= & 0 \mid (\nu c)\, P \mid P_1 \,|\, P_2 \mid {}^{l}b.P \mid {}^{l}c!t.P \mid !P \mid \\
 & \mid & {}^{l}\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(y) \colon P_1\ \mathsf{else}\ P_2 \\
b & ::= & c?x \mid \&_q(b_1, \cdots, b_n) \\
t & ::= & c \mid y \\
e & ::= & x \mid \mathsf{some}(t) \mid \mathsf{none}
\end{array}
$$

7.1 The Value-Passing Quality Calculus

In the following, we shall rely on a broadcast value-passing version of the Quality Calculus, namely, a subset of the calculus of Ch. 4 where channels are fixed names c as opposed to generic terms t. Thus, in this fragment of the calculus, a name received by an input cannot be used as a channel in the subsequent computation. Though the development of the protection analysis carries to a full-fledged version of the calculus (hence a quantitative counter-part of the availability analysis could in principle be devised), the price for the additional technicalities is not matched by major insights onto the idea, and thus we shall limit to comment upon such an extension in § 8.6.

7.1.1 Syntax and semantics

The Value-Passing Quality Calculus is displayed in Table 7.1, and consists of a fragment of the syntax of the Quality Calculus. In particular, channel positions in the broadcast $c!t$ and in the input $c?x$ are instantiated with plain names as opposed to terms (in particular, no variable y). Moreover, case clauses have now the main purpose of testing whether or not an input coming from a quality binder has been performed, and thus limit to check whether or not an optional data indeed contain data. Consistently with this simplification, terms t and expressions e do not include functions. For convenience in developing the analysis, recursive calls are now replaced by the replication operator $!P$.

As usual, in the following we consider closed processes (no free variable), and we make the simplifying assumption that processes are renamed apart so that names and variables are bound exactly once.

The semantics inherits some features of the basic calculus as well as the approach to broadcast communication of Ch. 6. The structural congruence $\equiv$ (cf. 4.2) is updated adding the classic rule for replication $!P \equiv P|!P$.

The transition relation inherits the two-layer structure of Table 6.4: we distinguish between local ($\xrightarrow{\alpha}$) and global transitions ($\Longrightarrow$), and we identify the semantics with the latter as dictated by rule (Sys) in Table 7.2, where $(\nu \overrightarrow{c})$ denotes the restriction of a list of names.

The transition relation simplifies Table 6.4 taking advantage of the restricted syntax and reasoning about replication. As terms and expressions contain no function symbol, the evaluations $t \rhd c$ and $e \rhd o$ are no more necessary: for the calculus directly applies substitutions to the continuations, terms and expressions in inputs, broadcasts, and case clauses have always been replaced with ground data or optional data when they are ready to execute. Quality binders are evaluated to Boolean as in the basic calculus, while the synchronisation relation is modelled after the one of Ch. 6 and is given in the last section of Table 7.2.

Restrictions are taken care of as in the previous chapter: no rule for transition under restriction is provided, therefore rule (Sys) must be applied to pull restrictions to the outer-most level. Rules (Par-) take care of interleaving. In particular, rule (Par-brd) takes care of interleaving broadcast, and applies in two situations.

First, if the parallel component P_2 is a replicated process, the broadcast has precedence over any action of instances of P_2. This means that when P_2 has the form $!c_1?x.P$, the number of input performed by its replicated instances will depend on the unfolding performed using the structural congruence as per rule (Sys). Note that this "freedom" issue does not arise in the semantics of Ch. 6.Observe, however, that the encoding of replication into recursion would lead to the same problem, since replication is essentially an unguarded recursion. The nodal point is that in the AQC recursive calls induce active τ transitions, while here replication is taken care of by structural rules. Finally, observe that a similar rule for unfolding replication in the semantics would lead to divergence due to rule (Par-tau).

It seems therefore that recursion matches more closely than replication our intuition of broadcast communication in presence of iterative or arbitrary long behaviour. Nonetheless, as far as the protection analysis presented in the following is concerned, we are only interested in *possible* behaviours, and therefore the more liberal replication operator is preferred as it admits all the behaviours recursion would allow and it is more convenient for specifying the analysis.

Second, if the parallel component P_2 is broadcast-prefixed, such a broadcast is delayed until the current one has exhausted its synchronisation opportunities. If none of the above applies, then either a synchronisation rule (In-) or an interleaving with a silent action (Par-tau) must take place.

7.1.2 Confidentiality labels

As argued in the introduction to this chapter, different points in a system have to be protected according to the value of the information they process. This idea can be formalised introducing a simple, non-functional extension to the calculus, where a program point of interest is instrumented with a *unique* label $l \in \mathcal{L}$. This is the case of input binders, outputs, and case clauses in Table 7.1. In the following, we will denote labels with the numerals $1, 2, \ldots$.

Table 7.2: A broadcast value-passing semantics with replication.

$$\frac{P_1 \equiv (\nu \overrightarrow{c})\, P_2 \quad P_2 \xrightarrow{\alpha} P_3}{P_1 \Longrightarrow P_3} \quad \text{(Sys)}$$

$${}^{l}c_1!c_2.P \xrightarrow{c_1!c_2} P \quad \text{(Brd)}$$

$$\frac{P_1 \xrightarrow{c_1!c_2} P_1' \quad c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{ff}} \theta}{P_1 \mid {}^{l}b.P_2 \xrightarrow{c_1!c_2} P_1' \mid {}^{l}b'.P_2} \quad \text{(In-ff)}$$

$$\frac{P_1 \xrightarrow{c_1!c_2} P_1' \quad c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{tt}} \theta}{P_1 \mid {}^{l}b.P_2 \xrightarrow{c_1!c_2} P_1' \mid P_2\theta} \quad \text{(In-tt)}$$

$${}^{l}\mathsf{case\ some}(c)\ \mathsf{of\ some}(y)\colon P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_1[c/y] \quad \text{(Then)}$$

$${}^{l}\mathsf{case\ none\ of\ some}(y)\colon P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_2 \quad \text{(Else)}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1|P_2 \xrightarrow{\tau} P_1'|P_2} \quad \text{(Par-tau)}$$

$$\frac{P_1 \xrightarrow{c_1!c_2} P_1'}{P_1|P_2 \xrightarrow{c_1!c_2} P_1'|P_2}\ \mathsf{if}\ P_2 = !P_2' \vee P_2 = {}^{l}c_1'!c_2'P_2' \quad \text{(Par-brd)}$$

$$c_1!c_2 \vdash c_1?x \to [\mathsf{some}(c_2)/x] \qquad c_1!c_2 \vdash c_3?x \to c_3?x \ \ \mathsf{if}\ c_1 \neq c_3$$

$$\frac{c_1!c_2 \vdash b_1 \to b_1' \quad \cdots \quad c_1!c_2 \vdash b_n \to b_n'}{c_1!c_2 \vdash \&_q(b_1, \ldots, b_n) \to \&_q(b_1', \ldots, b_n')}$$

We say that a label l is reached in an actual execution when the sub-process following l is ready to execute. Moreover, we assume to have a *confidentiality lattice* $(\Sigma = \{\sigma_1, \ldots, \sigma_n\}, \sqsubseteq_\Sigma)$, with greatest lower bound operator $\bigsqcap_\Sigma$, and a function $\mathsf{security} : \mathcal{L} \to \Sigma$ that maps labels into confidentiality levels. In particular, $\mathsf{security}(l_1) \sqsubseteq_\Sigma \mathsf{security}(l_2)$ denotes that the confidentiality (i.e., need for protection) of the program point indicated by l_2 is greater than or equal to the confidentiality of the program point indicated by l_1.

As an example of a confidentiality lattice, consider the *military lattice* given by $\Sigma = \{\mathsf{unclassified}, \mathsf{confidential}, \mathsf{secret}, \mathsf{top\text{-}secret}\}$, with the ordering $\mathsf{unclassified} \sqsubset_\Sigma \mathsf{confidential} \sqsubset_\Sigma \mathsf{secret} \sqsubset_\Sigma \mathsf{top\text{-}secret}$. More complex lattices, in particular non-linearly-ordered ones, are discussed in [Amo94, Ch. 7].

7.1.3 Security model

On top of the standard operational semantics of the calculus, our process-algebraic specifications rely on a security interpretation of communication actions. As binders are blocking actions, they can be thought of as security checks that require the knowledge of some information to be fulfilled. This knowledge is abstracted here by resorting to the notion of *secure channel* and disregarding the messages actually communicated, in line with the value-passing nature of the calculus. This idea is refined by quality binders: the existential quality guard $\exists$ describes scenarios in which different ways of fulfilling a check are available, e.g., different ways of proving one's identity. In contrast to this, the universal quality guard $\forall$ describes checks that require a number of sub-conditions to be met at the same time, and can be used to refine a security mechanism in terms of sub-checks.

On the other hand, an output represents the satisfaction of the security check specified by the corresponding channel. As the semantics is broadcast, all the input waiting on the given channel will be satisfied, that is, all the pending security checks will be fulfilled and the system will proceed until another blocking check is met. As a consequence, if the adversary can trigger a system component to make an output on a given channel c, it is as if c were under the control of the attacker, for all inputs on c in other components would be satisfied.

Finally, case clauses allow to inspect how a given security check, that is, a preceding binder, has been satisfied, by inspecting which input variables are bound to some values, that is, on which channels the communication took place.

As a result, an attack can be construed as a set of channels, namely, those channels needed to fulfil the security checks on a path to the target. We assume that a system P is deployed in a hostile environment, simulated by an adversary process Q running in parallel with P. The ultimate aim of the protection analysis is to compute what channels Q has to communicate over in order to drive P to a given location l, i.e., $P|Q \Longrightarrow^* C[^lP']$, where $C[^lP']$ denotes a sub-process of $P|Q$ that has reached label l and $\Longrightarrow^*$ denotes the reflexive and transitive closure of $\Longrightarrow$.

In order to compute the channels that Q needs to reach a program point l, this security model is translated into an attacker model where Q can *guess* every required channel, that is, whenever a security check on the way to l cannot be avoided, Q can fulfil it.

7.2 A Login System with Password Recovey

We demonstrate the channel-based approach of the framework on the problem of password recovery. As defined in the Common Weakness Enumeration [Mit], "it is common for an application to have a mechanism that provides a means

for a user to gain access to their account in the event they forget their password". It is then crucial to ensure that the protection to the account offered by the recovery mechanism is comparable to the protection provided by the password, otherwise we would have two paths leading to the same resource but performing a different amount of security checks. As noted by the Open Web Application Security Project [OSW]: "Weak password recovery processes allow stronger password authentication schemes to be bypassed. Non-existent password recovery processes are expensive to support or result in denial-of-service conditions."

We focus on a formalisations of the system such that authentication does not take place unless some interactions with the environment take place. In other words, we specify the security checks but not their fulfilment by the user who is supposed to be logging in, so as to rule out the legal way to authenticating and focus on malicious behaviours.

Below we encode a login system with possibility to recover a password in the Value-Passing Quality Calculus.

$$
\begin{aligned}
\mathsf{System} &\triangleq (\nu \mathsf{access})\,(\nu \mathsf{ok})\,(\nu \mathsf{pwd})\,(!(\mathsf{Login}|\mathsf{Recover})) \\
\mathsf{Login} &\triangleq {}^{1}\&_{\forall}(\mathsf{id}?x_{id}, \mathsf{pwd}?x_p).{}^{2}\mathsf{access!ok} \\
\mathsf{Recover} &\triangleq {}^{3}\&_{\forall}(\mathsf{id}?x'_{id}, \&_{\exists}(\mathsf{mail}?x_m, \mathsf{pin}?x_c)). \\
&\qquad {}^{4}\mathsf{case}\ x_m\ \mathsf{of}\ \mathsf{some}(y_m) : {}^{5}\mathsf{pwd!ok}\ \mathsf{else} \\
&\qquad\quad {}^{6}\mathsf{case}\ x_c\ \mathsf{of}\ \mathsf{some}(y_c) : {}^{7}\mathsf{pwd!ok}\ \mathsf{else}\ 0
\end{aligned}
$$

Process System is modelled after *Microsoft account* login mechanism, in charge of granting access to services such as mailboxes and technical forums. The main process is composed by two parallel sub-processes, running an unbounded number of times: the first one models the normal login procedure, while the second one abstracts the password recovery mechanisms.

According to process Login, in order to be granted access a user has to provide their own id and the corresponding password. This is mimicked by the two inputs expected by the quality binder at label 1, which are simultaneously active. The quality guard $\forall$ prescribes that both inputs have to be satisfied before proceeding, in any order. These inputs simulate two security checks: a party willing to authenticate into the system has to possess proper credentials, i.e., being able to communicate over id, pwd, and thus know such channels.

In the event a user forgot their password, the recovery mechanism comes into play. Microsoft offers two ways to recover a lost password: (i) a reset link is sent to an e-mail address previously registered by the user, or (ii) a 7-digit pin is sent to a phone number previously registered by the user. This behaviour is modelled by the quality binder at label 3 in process Recover. The binder is consumed as soon as the user has provided a valid id (e.g., an Outlook.com e-mail address), and proven their identity either through option (i) or option (ii). In the first case, the user needs to access a mailbox, simulated by an input on channel mail, while in the second case they have to provide the correct pin: the alternative is

implemented by the existential guard ∃ instrumenting the inner quality binder. The case clauses at labels 4, 6 determine what combination of inputs triggered passing the binder. In both cases, the user gets a valid password for the account in question, simulated by the outputs at label 5 and 7, and thus will be able to fulfil the check enforced by process Login.

Observe how case constructs are used to determine what combination allowed passing the binder: at label 4 we check whether the recovery process took place through another e-mail account, and if this is not the case then we check that the other condition, i.e., the phone challenge, is fulfilled. The main abstraction of our approach takes place at this level, as we can only test whether something is received on a given channel, but we cannot inspect the content of what is received. In other words, the knowledge of channel mail mimics the capability of accessing a given account, and thus we say that the semantic load of the communication protocol is shifted onto the notion of secure channel. Observe that this perspective seamlessly allows reasoning about the cost of attacking the system: to communicate over mail, an adversary has to get hold of a valid set of credentials, e.g., bribing the service provider or brute-forcing a cryptographic scheme, and this might prove more expensive than guessing the pin necessary to achieve authentication along the alternative path.

The key point of the example is indeed that no matter how strong a user's password is, an attacker can always try to guess a 7-digit sequence[1]. In particular, the requirement for a password is having at least 8 characters and containing different cases, numbers, or symbols, which (almost) automatically makes a password stronger than the pin! In terms of confidentiality levels, this suggests that the desired security architecture $\mathsf{security}(2) = \mathsf{security}(5) = \mathsf{security}(7)$ is not necessarily met, as the protection offered by the three paths leading to authenticating, i.e., to label 2, might not be uniform, depending on the cost notion we adopt. In the following, we shall demonstrate this violation relying on the simple security lattice $\mathsf{unrestricted} \sqsubset_\Sigma \mathsf{restricted}$, where the highest level is assigned to labels 2 and 4 to 7, while the lowest level is assigned to label 1 and 3, which represent public interfaces.

Whilst in this simple case a convincing conclusion can be drawn after careful investigation of the system, there is a general need to develop automated techniques to cope with more complex scenarios. In the remainder of the chapter we shall see how the protection analysis confirms the findings of our informal reasoning.

[1] As of Oct. 2013 there seemed to be no limit to the number of attempts one could try – we stopped our experiment at about 30. Some of our findings have been communicated in a number of situations and now we observe that they enforced both such limit and a daily threshold. Whilst this mutation does make it more difficult to quantify the strength of the mechanism (cf. § 7.4.2), it does not affect the relevance of the framework.

7.3 Discovering Attacks

The first challenge we tackle is discovering all the attacks leading to a given target. Given a process P modelling the system of interest and a label l in P identifying the target of the attack, the task of the analysis is to find all the sets of channels fulfilling the inputs occurring in P on the paths to l. In other words, we look for the knowledge that allows an adversary Q driving P to reach l.

To this end, P is translated into a set of propositional formulae (§ 7.3.1), termed *flow constraints*, describing how the knowledge of channels relates to reachability of locations, and how given a set of channels some other channels can be derived by the attacker. Then, the constraints are extended to consider the capabilities of the attacker (§ 7.3.2). Each model of the final set of constraints $P^{l}_{\Leftrightarrow}$ contains a set of channels that under-approximates the knowledge required to reach l.

7.3.1 From processes to flow constraints

The call $[\![P]\!]\mathsf{tt}$ of the recursive function $[\![\cdot]\!]\cdot$, defined in Table 7.3, translates a process P into a set of constraints of the form $\varphi \rightsquigarrow \overline{p}$, where φ is a propositional formula and $\overline{p}$ a positive literal. The intended semantics of a constraint states that if Q knows (enough information to satisfy) φ, then Q knows p, i.e., $\overline{p} = \mathsf{tt}$. As we shall see below, the antecedent φ accounts for the checks made on the path leading to disclosing p, namely input binders and case clauses. The consequent $\overline{p}$ can either stand for a channel literal $\overline{c}$, meaning that Q controls c, an input-variable literal $\overline{x}$, meaning that the attacker can satisfy the related input (i.e., $x = \mathsf{some}(c)$), or a label literal $\overline{l}$, meaning that l is reached.

At each step of the evaluation, the first parameter of $[\![\cdot]\!]\cdot$ corresponds to the sub-process of P that has still to be translated, while the second parameter is a logic formula, intuitively carrying the hypothesis on the knowledge Q needs to reach the current point in P. The translation function is structurally defined over processes as explained below.

If P is $\mathsf{0}$, then there is no location to be attained and thus no constraint is produced. If $P = !P'$ or $P = (\nu c)\, P'$, then it spontaneously evolves to P', hence Q does not need any knowledge to reach P' and gains no knowledge since no communication is performed. A parallel composition is translated taking the union of the sets into which the components are translated.

Communication actions have instead an impact on the knowledge of Q: inputs represent checks that require knowledge, outputs fulfil those checks, and case clauses determine the control flow. Assume that an action π is reached in the translation under hypothesis φ, that is, $[\![^{l}\pi.P']\!]\varphi$. Then, a constraint $\varphi \rightsquigarrow \overline{l}$ is generated: if the attacker fulfils the security checks on a path to l, then l is reached. In the logic interpretation of the constraints, this happens when φ evaluates to tt under a model given by the knowledge of the attacker,

forcing $\overline{l}$ to be tt (as standard implication would do). Moreover, the nature of action π determines whether or not other constraints are produced and how the translation proceeds.

Consider a simple input ${}^{l}c?x.P'$: whenever the action is consumed, it must be that the attacker controls the communication channel c, hence we translate P' under the hypothesis $\varphi \wedge \overline{c}$. Moreover, if the input is consumed, then x must be bound to $\mathsf{some}(c')$, hence we produce a constraint $(\varphi \wedge \overline{c}) \rightsquigarrow \overline{x}$. These two steps respectively accommodate the hypothesis we need for passing a binder (success condition), and the conclusion we can establish whenever a binder is passed (strongest post-condition), in analogy with the availability analysis of § 5.4. In Table 7.3 functions hp and th take care of formalising this intuition, that seamlessly applies to quality binders, where the hypothesis is augmented accounting for the combinations of inputs that satisfy the binder, as dictated by the quality guard q. The last section of Table 7.3 shows two cases for q, but any Boolean predicate can be used.

Note the profound symmetry between the precondition generated by the protection analysis for a label l and the formula that the robustness analysis of Ch. 4.6 would attach to the same program point. The combination of input variables of the latter is essentially replaced in the former by the combination of channels over which those input variables must be received.

The execution of an output ${}^{l}c!t$ satisfies all the security checks represented by inputs waiting on c. Therefore, if Q can trigger such output, it obtains the knowledge related to c without having to know the channel directly, and thus a constraint $\varphi \rightsquigarrow \overline{c}$ is generated. It is worthwhile observing that this behaviour is justified by the broadcast semantics, and by the fact that the calculus is limited to testing whether or not something has been received over a given channel, shifting the semantic load on the notion of secure channel. Moreover, note that the asymmetry between input and output is due to the fact that outputs are non-blocking.

A case construct is translated by taking the union of the constraints into which the two branches are translated: as the check is governed by the content of the case variable x, we record that the then branch is followed only when x is bound to $\mathsf{some}(c)$ by adding a literal $\overline{x}$ to the hypothesis, as we do for inputs, and we add $\neg\overline{x}$ if the else branch is followed.

The set of constraints $[\![P]\!]\mathsf{tt}$ computed according to Table 7.3 can be normalised so as to produce a compact representation of P. Whenever two rules $\varphi \rightsquigarrow \overline{p}$ and $\varphi' \rightsquigarrow \overline{p}$ are in $[\![P]\!]\mathsf{tt}$, they are replaced with a single rule $(\varphi \vee \varphi') \rightsquigarrow \overline{p}$. This simplification is intuitively sound for if φ leads to obtain p and φ' leads to obtain p, then p is available to the attacker under the condition that $\varphi \vee \varphi'$ is known. In the following, we assume to deal with sets of constraints in such format.

Table 7.3: The translation $[\![P]\!]\mathsf{tt}$ from processes to flow constraints.

$$
\begin{array}{lcl}
[\![0]\!]\varphi & = & \emptyset \\
[\![!P]\!]\varphi & = & [\![P]\!]\varphi \\
[\![P_1|P_2]\!]\varphi & = & [\![P_1]\!]\varphi \cup [\![P_2]\!]\varphi \\
[\![(\nu c)\,P]\!]\varphi & = & [\![P]\!]\varphi \\
[\![^l b.P]\!]\varphi & = & [\![P]\!](\varphi \wedge \mathsf{hp}(b)) \cup \mathsf{th}(\varphi, b) \cup \{\varphi \rightsquigarrow \bar{l}\} \\
[\![^l c!t.P]\!]\varphi & = & [\![P]\!]\varphi \cup \{\varphi \rightsquigarrow \overline{c}\} \cup \{\varphi \rightsquigarrow \bar{l}\}
\end{array}
$$

$$[\![^l \mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2]\!]\varphi = [\![P_1]\!](\varphi \wedge \overline{x}) \cup [\![P_2]\!](\varphi \wedge \neg\overline{x}) \cup \{\varphi \rightsquigarrow \bar{l}\}$$

$$\mathsf{hp}(c?x) = \overline{c} \qquad \mathsf{hp}(\&_q(b_1, \ldots, b_n)) = [\![q]\!](\mathsf{hp}(b_1), \ldots, \mathsf{hp}(b_n))$$

$$\mathsf{th}(\varphi, c?x) = \{(\varphi \wedge \overline{c}) \rightsquigarrow \overline{x}\} \qquad \mathsf{th}(\varphi, \&_q(b_1, \ldots, b_n)) = \bigcup_{i=1}^{n} \mathsf{th}(\varphi, b_i)$$

$$[\![\forall]\!](\overline{c_1}, \ldots, \overline{c_n}) = \bigwedge_{i=1}^{n} \overline{c_i} \qquad [\![\exists]\!](\overline{c_1}, \ldots, \overline{c_n}) = \bigvee_{i=1}^{n} \overline{c_i}$$

7.3.2 Modelling the attacker

A rule $\varphi \rightsquigarrow \overline{p}$ in $[\![P]\!]\mathsf{tt}$ describes how Q can attain p playing according to the rules of the system, namely fulfilling the checks described by φ. Nonetheless, when p is a channel, an attacker can always try to obtain it directly, for instance guessing some cryptographic keys or bursting a gate. In order to account for this possibility, we enrich each rule $\varphi \rightsquigarrow \overline{c}$ by replacing the antecedent with the disjunction $g_c \vee \varphi$, where literal g_c (for "guess c") represents the possibility of learning c directly. For each channel c such that no rule $\varphi \rightsquigarrow \overline{c}$ is in $[\![P]\!]\mathsf{tt}$, we add to $[\![P]\!]\mathsf{tt}$ a constraint $g_c \rightsquigarrow \overline{c}$, expressing that Q has no option but guessing the channel.

Finally, observe that having added the literals g_c, which tell how the attacker can get hold of a channel in any other way than those legal in P, interpreting the relation $\rightsquigarrow$ as the propositional bi-implication $\Leftrightarrow$ preserves the minimal models of the system of constraints. A constraint $\varphi \vee g_c \Leftrightarrow \overline{c}$ states that c is only obtained by guessing or by making P disclose it and, on the other hand, that if c is known to the attacker it must be because they have guessed it or because they made P disclose it.

In the following, given a label l of interests, we shall write $P^l_{\Leftrightarrow}$ to denote the conjunction of constraints $[\![P]\!]\mathsf{tt}$ which have undergone the transformations mentioned above. In particular, in $P^l_{\Leftrightarrow}$

- $\bar{l}$ is a fact, expressing the query we want to study:

- there is exactly one conjunct $\varphi \Leftrightarrow \overline{l}$;
- for each channel c occurring in P, there is exactly one conjunct $\varphi \Leftrightarrow \overline{c}$, and φ has the form $g_c \vee \varphi'$ where g_c does not occur elsewhere in $P^l_{\Leftrightarrow}$;

Intuitively, we assume that l is reached and we look for the consequences in terms of truth values of channel literals (i.e., we look for *implicants* of $\overline{l}$ [DM94]). In the following, we present a SAT-based solution to this problem.

Translating the example login system. The translation of the log-in system of § 7.2 returns the following flow constraints, augmented as explained above. For the sake of simplicity we omit the occurrences of tt as a conjunct in all left-hand sides.

$$\left.\begin{array}{l} \overline{1} \\ \overline{\mathsf{id}} \Leftrightarrow \overline{x_{id}} \\ \overline{\mathsf{pwd}} \Leftrightarrow \overline{x_p} \\ \overline{\mathsf{id}} \wedge \overline{\mathsf{pwd}} \Leftrightarrow \overline{2} g_{\mathsf{access}} \vee (\overline{\mathsf{id}} \wedge \overline{\mathsf{pwd}}) \Leftrightarrow \overline{\mathsf{access}} \end{array}\right\} \text{ from } [\![\mathsf{Login}]\!]\mathsf{tt}$$

$$\left.\begin{array}{l} \overline{3} \\ \overline{\mathsf{id}} \Leftrightarrow \overline{x'_{id}} \\ \overline{\mathsf{mail}} \Leftrightarrow \overline{x_m} \\ \overline{\mathsf{pin}} \Leftrightarrow \overline{x_c} \\ \underbrace{\overline{\mathsf{id}} \wedge (\overline{\mathsf{mail}} \vee \overline{\mathsf{pin}})}_{\varphi} \Leftrightarrow \overline{4} \\ \varphi \wedge \overline{x_m} \Leftrightarrow \overline{5} \\ \varphi \wedge \neg\overline{x_m} \Leftrightarrow \overline{6}\varphi \wedge (\neg\overline{x_m}) \wedge \overline{x_c} \Leftrightarrow \overline{7} \\ g_{\mathsf{id}} \Leftrightarrow \overline{\mathsf{id}} \\ g_{\mathsf{mail}} \Leftrightarrow \overline{\mathsf{mail}} \\ g_{\mathsf{pin}} \Leftrightarrow \overline{\mathsf{pin}} \\ g_{\mathsf{pwd}} \vee (\varphi \wedge \overline{x_m}) \vee (\varphi \wedge (\neg\overline{x_m}) \wedge \overline{x_c}) \Leftrightarrow \overline{\mathsf{pwd}} \end{array}\right\} \text{ from } [\![\mathsf{Recover}]\!]\mathsf{tt}$$

where the only way for Q to know $\mathsf{id}, \mathsf{mail}$, pin is to guess them. Observe that the capability of using the password channel (last constraint) is obtained either by satisfying the recovery mechanism or by guessing pwd.

7.3.3 A SAT-based solution technique

A model of $P^l_{\Leftrightarrow}$ can be represented as a function mapping the literals in $P^l_{\Leftrightarrow}$, denoted by $\mathsf{dom}(\mu)$, to truth values $\{\mathsf{ff}, \mathsf{tt}\}$. Now, denoted by *Names* the set of names (channels) occurring in the process P under study, then the set

$$\mathsf{attack}(\mu) = \{\overline{c} \in \mathsf{dom}(\mu) \mid c \in \mathit{Names} \wedge \mu(g_c) = \mathsf{tt}\}$$

identifies a set of channels that, if guessed, satisfies the constraints in $P^l_{\Leftrightarrow}$. In the semantics, $\mathsf{attack}(\mu)$ under-approximates a set of channels that fulfil the

security checks on a path to l in P, i.e., a way for Q to drive P to l. Denoted by M^l the set of all models of $P^l_{\Leftrightarrow}$, the corresponding set of sets of channels $\mathsf{attack}(M^l)$, obtained by point-wise application of attack to the elements of M^l, contains under-approximations to all the attacks leading to l.

Hence, in order to solve the analysis we need essentially to compute all the models M^l of the propositional constraints $P^l_{\Leftrightarrow}$, that is, we have to solve the ALL-SAT problem for the input formula $P^l_{\Leftrightarrow}$. If no solution is found, i.e., $P^l_{\Leftrightarrow}$ is unsatisfiable, then the program point indicated by l is not reachable.

It is worthwhile observing that the translation into flow constraints over-approximates the behaviour of a process, giving rise to more executions than those actually arising in the semantics. Such spurious executions correspond to attacks that under-approximate the sets of channels required to reach the target l, and therefore the overall analysis results in an under-approximation. This intuition is formalised in the following correctness statement:

$$\text{if} \quad P|Q \Longrightarrow^* C[^l P'] \quad \text{then} \quad \exists \mathcal{N} \in \mathsf{attack}(M^l)\,.\,\mathcal{N} \subseteq \mathsf{fc}(Q)$$

i.e., for all the executions in which Q can drive P to l, the analysis computes a set of channels $\mathcal{N} \in \mathit{Names}$ that under-approximates the knowledge required of Q. The formulation of the actual theorem requires to establish some additional notation, and therefore is deferred to Appendix C.1. In the following, we shall focus instead on an example process that pinpoints the imprecision of the analysis.

A main source of over-approximation in the translation to flow constraints is the treatment of replication and restriction, whose interplay is simply disregarded by the analysis. As a matter of fact, a name restricted under replication is a different name in all the instances of the replicated process. Consider the following example:

$$P \triangleq\ (\nu c)\,((!(\nu a)\,{}^{1}a?x_a.{}^{2}c!c)\,|{}^{3}c?x_c.{}^{4}c?x'_c.{}^{5}\ldots)$$

Let label 5 be the location of interest and consider the following translation of P into flow constraints, conveniently simplified for the sake of conciseness:

$$(g_a \Leftrightarrow \overline{a})\ \wedge\ (g_c \vee \overline{a} \Leftrightarrow \overline{c})\ \wedge\ (\overline{c} \Leftrightarrow \overline{5})\ \wedge\ (\mathsf{tt} \Leftrightarrow \overline{5})$$

According to the translation, an attacker can reach label 5 either by knowing a or by knowing c, that is, $\mathsf{attack}(M^5) = \{\{a\}, \{c\}, \{a, c\}\}$.

Whilst it is clear that if the attacker guesses c then the security checks at labels $3, 4$ can be satisfied, it is less obvious what it means for the attacker to guess a. In fact, if the adversary makes an output on a twice, then two outputs on c are triggered, and thus the checks on the path to the goal 5 are fulfilled. According to the semantics this may happen in all traces in which the replication is unfolded at least twice. Nonetheless, the two instances of a in the two copies of the process are different: α-renaming applies producing names a and a'. Hence, claiming that 5 can be reached by guessing a, the analysis under-approximates the knowledge required of the attacker, who needs a and a'.

7.4 Quantifying Attacks

In the previous section we have presented a SAT-based solution technique, where each model of $P^{l}_{\Leftrightarrow}$ contains a set of channels that are necessary to fulfil an attack. Not all security mechanisms, however, offer the same protection guarantees, that is, not all channels are equal. A retinal scan can prove more difficult to bypass than a pin lock, and thus offer more protection. This is not the case, however, of an insider who is authorised to enter the corresponding room. A cost structure over channels facilitates formalising these considerations, and assigning costs to channels naturally leads to quantify sets of channels, that is, attacks.

The characterisation of attacks in terms of cost allows to order them, and ultimately to focus on those which are deemed the most likely given our understanding of the candidate attacker profiles. Moreover, the *minimal* cost of reaching a given location l identifies the protection deployed to guard l in the implementation, which can be contrasted with the desiderata of the specification. The higher the cost for the attacker, the higher the protection guarding the target.

In the following, we extend the qualitative analysis of § 7.3 to a quantitative setting. In particular, we should follow a modular approach according to which cost considerations are developed on top of the structure of the original analysis. The benefit of a layered strategy is two-fold: on the one hand, we present a technique that can be exploited to transform a great many qualitative analyses into quantitative analyses; on the other hand, whenever quantitative information about the entities in question is not available, we can resort to the qualitative solution.

7.4.1 From qualitative to quantitative considerations

Let cost be a function from channels $c \in \mathit{Names}$ to costs $k \in \mathcal{K}$. Formally, we require $(\mathcal{K}, \oplus)$ to be a commutative monoid (also known as Abelian monoid), that is, $\oplus$ is an associative and commutative binary operation on the set $\mathcal{K}$ and has an identity element. Moreover, we require $\mathcal{K}$ to be equipped with a partial order $\sqsubseteq_{\mathcal{K}}$, such that $(\mathcal{K}, \sqsubseteq_{\mathcal{K}})$ is a lattice, and $\oplus$ to be extensive, that is, the sum of two elements always dominates both the summands:

$$\forall k_1, k_2 \in \mathcal{K} \,.\, k_1 \sqsubseteq_{\mathcal{K}} (k_1 \oplus k_2) \;\wedge\; k_2 \sqsubseteq_{\mathcal{K}} (k_1 \oplus k_2) \tag{7.1}$$

Finally, we assume that $\oplus$ is monotone and the least element $\bot \in \mathcal{K}$ is its identity element, that is, $\oplus$ is an upper bound operator of the lattice $(\mathcal{K}, \sqsubseteq_{\mathcal{K}})$, and therefore satisfies condition (7.1). For the sake of simplicity, we assume that the costs of channels are independent.

For the sake of simplifying the notation, in the following we shall feel free to apply the function cost to sets of names, according to the following definition:

$$\begin{aligned} &\mathsf{cost} : \mathcal{P}(\mathit{Names}) \to \mathcal{K} \\ &\mathsf{cost}(\{c_1, \ldots, c_n\}) = \textstyle\bigoplus_{i=1}^{n} \mathsf{cost}(c_i) \end{aligned}$$

Likewise, we extend the function cost also to sets of sets of names by point-wise application:

$$\mathsf{cost} : \mathcal{P}(\mathcal{P}(Names)) \to \mathcal{P}(\mathcal{K})$$

$$\mathsf{cost}(\{c_1^1, \ldots, c_{n_1}^1\}, \ldots, \{c_1^m, \ldots, c_{n_m}^m\}) = \\ \big\{\mathsf{cost}(\{c_1^1, \ldots, c_{n_1}^1\}), \ldots, \mathsf{cost}(\{c_1^m, \ldots, c_{n_m}^m\})\big\}$$

Therefore, given a model μ of $P^l_{\Leftrightarrow}$, the corresponding set $\mathsf{attack}(\mu) = \{c_1, \ldots, c_n\}$ can be quantified as

$$\mathsf{cost}(\mathsf{attack}(\mu)) = \bigoplus_{c \in attack(\mu)} \mathsf{cost}(c)$$

As we mentioned above, however, a conservative approach to security would consider the attacks of *minimal* cost. Hence, given two attacks, i.e., two distinct models μ, μ', we would discard μ' in case $\mathsf{cost}(\mathsf{attack}(\mu)) \sqsubset_{\mathcal{K}} \mathsf{cost}(\mathsf{attack}(\mu'))$. We can thus restrict the set of models M^l to the ones bearing attacks of minimal cost:

$$\mathsf{minimal}(M^l) = \big\{\mu \in M^l \,|\, \forall \mu' \in M^l.\mathsf{cost}(\mathsf{attack}(\mu')) \not\sqsubset_{\mathcal{K}} \mathsf{cost}(\mathsf{attack}(\mu))\big\}$$

It is worthwhile noticing that $\mathsf{minimal}(M^l)$ may contain more than one model, as (i) we consider all the attacks with same cost and (ii) some attacks may have incomparable costs in case the cost set is not linearly ordered.

Now, we can relate an attack to the corresponding security level $\sigma \in \Sigma$ required to counter it by means of a function $\mathsf{level} : \mathcal{K} \to \Sigma$, compressing cost regions into security levels:

$$\mathsf{level}(k) = \begin{cases} \sigma_1 & \text{if } k \in \{k_1^1, \ldots, k_{h_1}^1\} \\ \vdots & \\ \sigma_m & \text{if } k \in \{k_1^m, \ldots, k_{h_m}^m\} \end{cases}$$

where level is a well-defined function if the sets of costs $\{k_1^i, \ldots, k_{h_i}^i\}$ are pair-wise disjoint and their union is $\mathcal{K}$. Moreover, it is natural to require that level is monotone. A simple example in the cost set $(\mathbb{N}, +)$ and security lattice $\mathsf{low} \sqsubset \mathsf{medium} \sqsubset \mathsf{high}$ is given by the choice

$$\mathsf{level}(k) = \begin{cases} \mathsf{low} & \text{if } k \leq 1024 \\ \mathsf{medium} & \text{if } 1024 < k \leq 2048 \\ \mathsf{high} & \text{if } 2048 < k \end{cases}$$

where numbers could represent the length of cryptographic keys, and we state for instance that a program point is poorly protected if no more than 1024 bits are necessary to attain it (for a fixed cryptosystem).

Finally, we extend level to work on sets of costs so as to encompass all the sets of channels produced by the analysis at once:

$$\begin{aligned}&\mathsf{level} : \mathcal{P}(\mathcal{K}) \to \Sigma\\ &\mathsf{level}(\{k_1, \ldots, k_n\}) = \{\mathsf{level}(k_1), \ldots, \mathsf{level}(k_n)\}\end{aligned}$$

where the input $\{k_1, \ldots, k_n\}$ is the set of costs of all minimal attacks, computed as

$$\mathsf{cost}(\mathsf{attack}(\mathsf{minimal}(M^l)))$$

Finally, the greatest lower bound $\bigsqcap_\Sigma$ is used to derive the greatest security level compatible with all attacks in $\mathsf{minimal}(M^l)$, that is, the protection of a program point corresponds at most to the cost of the weakest path leading to it.

A graphical illustration of the various components of the analysis is displayed in Fig. 7.1, where it is apparent how the quantitative analysis is built on top of the qualitative analysis. Intuitively, function $\mathsf{security}$ is the *specification* expressing the target security architecture of a system with respect to a given security lattice, while $\mathsf{level}(\mathsf{cost}(\mathsf{attack}(\mathsf{minimal}(M^l))))$ captures (an under-approximation of) how this architecture has been realised in the *implementation*.

The overall aim of the analysis, i.e., checking whether the deployed protection lives up to the required confidentiality, can thus be expressed by the property

$$\forall l \in \mathcal{L} \,.\, \mathsf{security}(l) \sqsubseteq_\Sigma \mathsf{level}(\mathsf{cost}(\mathsf{attack}(\mathsf{minimal}(M^l))))$$

A violation of this condition is referred to as a potential *inversion of protection*. The overall under-approximation of the analysis is the results of minimising over the costs of under-approximating sets of channels, as we we have seen for the qualitative analysis in § 7.3.3.

Turning our attention again to availability considerations, it is worthwhile observing that the protection analysis can be used to quantify the minimal cost an attacker incur to cause DoS, thereby *validating* cost-based proactice strategies as those surveyed in §3

7.4.2 Optimisation Modulo Theories

In order to compute the set of sets of channels $\mathsf{attack}(\mathsf{minimal}(M^l))$ that allow reaching l incurring minimal costs, we need to solve an optimisation problem subject to the Boolean constraints $P^l_\Leftrightarrow$. There exist various techniques to cope with such problems, each suitable for particular choices of cost sets and objective functions. One solution is to first compute M^l and then minimise it by comparing models as explained above. Nonetheless, cost information can be levered to skip non-optimal models during the search, hence improving the performance. In the following, we show how to exploit an SMT solver to tackle the problem in its most general form. We limit to mention that linear programming techniques such as Pseudo-Boolean optimisation [BH02] are efficient alternatives for dealing with the monoid $(\mathbb{Z}, +)$ and linear objective functions.

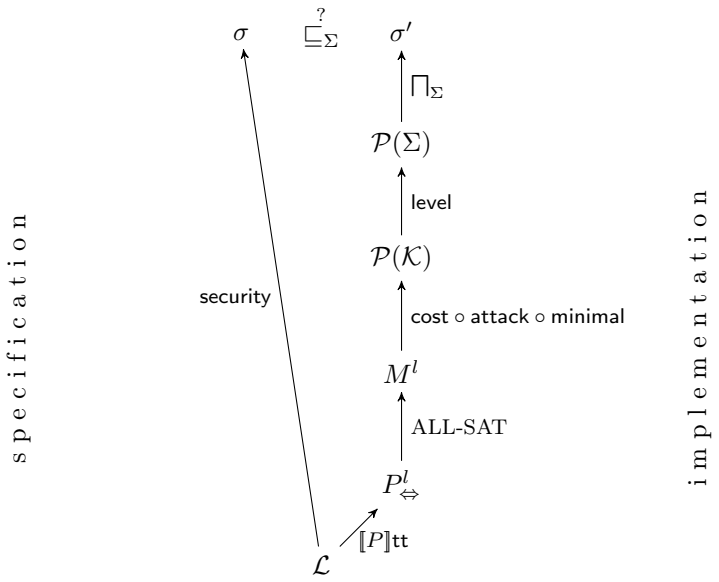


Figure 7.1: The quantitative protection analysis at a glance.

An SMT-based solution. In a nutshell, our task reduces to compute models of $P^l_\Leftrightarrow$ (containing attacks) of minimal cost in the lattice $\mathcal{K}$. In other words, we are looking for *prime* implicants of $\bar{l}$ [DM94], where primality is sought with respect to the given cost set.

Such an optimisation problem can be tackled by computing models for a list $\Pi_1, \ldots, \Pi_n$ of SMT problems, where Π_i is a more constrained version of Π_{i-1} that requires to improve on the cost of the current solution. The initial problem Π_1 consists of the propositional constraints $P^l_\Leftrightarrow$ and of the objective function, whose value on the current model is stored in variable goal.

The objective function is essentially the cost of the current model. In order to compute cost(attack(μ)) into variable goal as part of μ itself we define:

$$\mathsf{goal} := \bigoplus_{i=1}^{n} (\mathsf{if}\ g_{c_i}\ \mathsf{then}\ \mathsf{cost}(c_i)\ \mathsf{else}\ \bot)$$

where we combine the costs of all the channels that must be guessed, that is, the channels c_i's such that the corresponding guessing literal g_{c_i} is found to be tt. Otherwise, if a g_{c_i} is ff, then the corresponding c_i needs not be guessed and its cost does not contribute to the cost of the attack. Recall that the least element $\bot$ of the cost lattice $\mathcal{K}$ does not contribute any cost, for it is the neutral element with respect to the cost combinator $\oplus$. Hence, by construction we have $\mu(\mathsf{goal}) = \mathsf{cost}(\mathsf{attack}(\mu))$.

Data: The problem $\Pi \triangleq P^l_{\Leftrightarrow} \wedge (\mathsf{goal} := \bigoplus_{i=1}^{n}(\mathsf{if}\ g_{c_i}\ \mathsf{then}\ \mathsf{cost}(c_i)\ \mathsf{else}\ \bot))$
Result: the set $\mathcal{M}$ of pairs $(\mu, \mathsf{cost}(\mathsf{attack}(\mu)))$ such that
$\mu \in \mathsf{minimal}(M^l)$
$\mathcal{M} \leftarrow \emptyset$;
while Π *satisfiable* **do**
  $\mu \leftarrow \mathsf{get\text{-}model}(\Pi)$;
  $k \leftarrow \mu(\mathsf{goal})$;
  forall the $(\mu', k') \in \mathcal{M} \mid k \sqsubset_{\mathcal{K}} k'$ **do** // μ `outperforms` μ'
  | $\mathcal{M} \leftarrow \mathcal{M} \setminus \{(\mu', k')\}$
  end
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{(\mu, k)\}$;
  $\Pi \leftarrow \Pi \wedge \neg(\bigwedge_{i=1}^{n}(\overline{c_i} = \mu(\overline{c_i}))) \wedge \neg(\mathsf{goal} \sqsupset_{\mathcal{K}} k)$;
end

Algorithm 1: The SMT-based solution procedure.

Then, while the problem is satisfiable, we improve on the cost of the current model by asserting new constraints which tighten the value of goal, until unsatisfiability is reported. Algorithm 1 displays the pseudo-code of the procedure. In particular, observe that when a new problem Π_i is generated, additional constraints are asserted that ask for (i) a different model and (ii) a non-greater cost: the former condition speeds up the search, while the latter explores the cost frontier.

The termination of the algorithm is ensured by the finiteness of possible models to the propositional variables of the Π_i's, whose propositional structure does not change throughout the loop, and by the fact that the same model cannot occur twice as solution due to the new constraints we generate in each iteration. At most, we need to solve as many Π_i's as there are models of $P^l_{\Leftrightarrow}$, which coincide with the qualitative analysis (ALL-SAT). The correctness of the procedure stems from the fact that when unsatisfiability is claimed, by construction of the Π_i's there cannot exist further models that comply with the cost constraints.

It is worthwhile noticing how resorting to propositional logic integrates with the overall under-approximating nature of the analysis: a channel can either be learnt or not, and its cost contribute or not to the cost of an attack. This means that we do not keep track of the number of attempts made to guess some information, and always assume that guessing c is successful whenever g_c is found to be true. In order words, for a fixed cost set, we are considering the luckiest or cleverest attacker. As for the cost set (comparing and combining costs), SMT solvers offer native support for numeric costs and common mathematical functions, while more complex cost sets have to be encoded manually.

Finally, observe that the procedure above is not dependent on our analysis, but can be generally exploited to find optimal models of arbitrary logic formulae

and for arbitrary cost sets, and can be seamlessly extended to more complex logics.

Attacking the example login system. Consider the login system discussed in § 7.2, and assume to work in the cost set $(\mathbb{N}, +)$. The engineering of a sensible cost map is a delicate task that involves cryptographic arguments and falls outside the scope of this dissertation. For the time being let us make the simplistic assumption that costs to channels are given by the number of bits to be guessed: $\mathsf{cost}(\mathsf{pin}) = 28$ (7 digits, 4 bits each) and $\mathsf{cost}(\mathsf{pwd}) = 56$ (8 symbols, 7 bit per ASCII symbol). We assume that the user id is known (the target e-mail itself, for instance), and that the password of the third-party mailbox is comparable to pwd (as the constraints put by Microsoft are customary also to other e-mail providers). Finally, we disregard channel access as it is only used after the label of interest is reached.

The problem is thus to minimise

$$\sum_{c \in Names} (\mathsf{if}\ g_c\ \mathsf{then}\ \mathsf{cost}(c)\ \mathsf{else}\ 0)$$

under the constraints given by $P^2_{\Leftrightarrow}$. Instructed with this input, our procedure finds that the formula is satisfiable and the single cheapest model μ contains the assignments

$$g_{\mathsf{id}} \mapsto \mathsf{tt} \qquad g_{\mathsf{pwd}} \mapsto \mathsf{ff} \qquad g_{\mathsf{mail}} \mapsto \mathsf{ff} \qquad g_{\mathsf{pin}} \mapsto \mathsf{tt}$$

with cost given by $\mathsf{cost}(\mathsf{id}) + \mathsf{cost}(\mathsf{pin}) = 28$, and entailing $\mathsf{attack}(\mathsf{minimal}(P^2_{\Leftrightarrow})) = \{\{\mathsf{id}, \mathsf{pin}\}\}$.

As for the desired security levels, we observed in § 7.1 that $\mathsf{security}(2) = \mathsf{security}(5) = \mathsf{security}(7) = \mathsf{restricted}$ should hold, for we want the account to be equally protected on all the paths leading to granting access. As the cost of accessing an account in normal condition is 56, it is reasonable to set

$$\alpha(k) = \begin{cases} \mathsf{restricted} & \mathsf{if}\ k \geq 56 \\ \mathsf{unrestricted} & \mathsf{otherwise} \end{cases}$$

We would like to verify that $\mathsf{restricted} \sqsubseteq_\Sigma \bigsqcap_\Sigma \mathsf{level}(28)$, which is false. Hence, it is the case that the implementation potentially guarantees less protection than the amount required by the specification, and therefore we shall issue a warning to the designer of the system.

Obviously, costs are central in determining the outcome of the analysis. For example, we could consider to choose pwd from a password dictionary, for it is unrealistic to assume randomly generated bits: the size of such dictionaries is usually less than the number of sequences of 7 digits, and thus with this cost map the analysis might say that the recovery mechanism offers enough protection.

Likewise, a limit to the number of attempts – as later enforced by Microsoft on its platform – can equalise the protection of the two mechanisms or make the one offered by the recovery procedure even stronger.

Finally, it is worthwhile noticing that the framework can be exploited to measure the *distance* between the implementation and the specification, and not only their qualitative compliance.

A semantic interpretation of guessing. It would be possible to formulate a neat semantic interpretation of the guessing capability of the attacker. Assume to deal with processes of the form $((\nu\overrightarrow{c})\,P)|Q$, where the first component is the system under study, in which all restrictions are at the outer-most level, and Q is the attacker. Now, for Q to interact with P, the attacker needs to move inside the scope of some restrictions so as to share some channel names with P. Whenever Q enters the scope of a restriction (νc), the name c is guessed. In order to account for the cost $k \in \mathcal{K}$ of guessing a name, we can instrument each restriction with the corresponding cost, writing $(\nu^k c)$, and then augment the scope extension rule (New3) of Table 4.2 so as to accumulate the cost of names that are guessed. The standard semantics of restriction used in security applications of process calculi, according to which a new name c is only known to legal participants unless leaked (P, in our case), is encompassed by assigning c an infinite cost.

Though possible, such an extension of the semantics is not necessary to prove the correctness of the analysis. Every assignment that satisfies the propositional constraints leads Q to reach the location l of interest, hence also the ones of minimal costs. Nonetheless, it is worthwhile noticing that such a quantitative point of view on restrictions generalises the distinction between the operators new and hide introduced in the *secret* π-calculus [GPV12]. The operator $\mathsf{hide}\,c$, which introduces a name c inhibiting its scope extension, would correspond to $(\nu^\infty c)$, while we would have a more fine-grained view on plain scope extension.

7.4.3 Complex cost structures

So far we have worked with an example in the cost set $(\mathbb{Q}, +)$, for it is natively encoded into SMT solvers and matches a first intuition of the notion of cost. Nonetheless, it is often difficult to provide an absolute estimate of the strength of a protection mechanism: sometimes different mechanisms are even incomparable, as cryptography and physical security might be. In such cases, it is more natural to describe the relative strength of a set of mechanisms with respect to each other. This is achieved by computing the analysis over symbolic and partially-ordered cost structures.

As a basic example, consider the cost lattice displayed in Fig. 7.2: we could characterise the cost of obtaining given information as cheap, if it does not require a specific effort, as cpu, if it requires significant computational capabilities

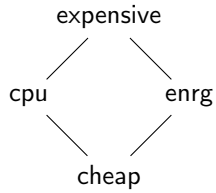


Figure 7.2: The Hasse diagram of a partially-ordered cost structure.

(e.g., breaking an encryption scheme), as enrg, if it requires to spend a considerable amount of energy (e.g., engaging in the wireless exchange of a number of messages), or as expensive, if it requires both computations and energy. In order to combine such costs, a suitable choice is to take as monoid operator $\oplus$ the least upper bound $\sqcup$ of two elements in the cost lattice.

Observe that Algorithm 1 is already equipped to cope with the general problem of optimising on partially-ordered cost sets. An interesting case of non-linear cost sets is offered by the study of security in Cyber-Physical Systems, where components combine both software and physical features [Vig12]. In particular, in such systems an attack could require to assemble cyber actions with physical tampering, whose costs can either be comparable or not depending on the nature of the quantities we are interested in (for instance, energy and memory are not directly comparable).

In conclusion, three elements push independently for the comprehensive SMT-based approach: the non-linearity of the cost set, its symbolic nature, and the non-linearity of the objective function.

A mundane approach to password recovery. *Yahoo!Mail* password recovery mechanism differs from the one provided by *Microsoft account* in that it is (also) possible to recover a password by answering two personal questions chosen upon the registration of the mailbox. The user needs hence to provide an id and to answer two questions like "What is your mother's maiden name?". It is unclear how to quantify the difficulty of such questions in terms of numbers. Nonetheless, a substantial consensus of opinion sustains the feeling that it is simpler for an attacker to get hold of such secrets than guessing a randomly generated pin [GJ05] – often the answers to such questions can be uncovered exploiting social media [Rab08]

A symbolic quantification of the two paths leading to logging into the mailbox can then be modelled in the cost set $(\{\mathsf{cheap}, \mathsf{expensive}\}, \oplus)$, where the cost of answering a question is cheap and the cost of guessing the password is expensive. As for the monoid operator, a suggestion is to use (the least-upper bound) $\mathtt{max}(\cdot, \cdot)$, since asking one, two, or three questions will annoy an attacker but does not really make their task much harder.

The SMT implementation of this example requires to encode the cost struc-

ture, that is, declaring its elements and defining the ordering relation as well as the monoid operator. Running the analysis in this cost set, we verified the existence of a cheap path leading to authenticating, namely the path that exploits the question-based recovery mechanism. The framework thus suggests that such an option should not be offered, otherwise we would not provide uniform security on all paths guarding the protected region.

7.5 The Quality Tool

A proof-of-concept implementation of the framework is available in Java at

www.imm.dtu.dk/~rvig/quality-tool.html

The *Quality Tool* takes as input an ASCII representation of a Value-Passing Quality Calculus process P and generates the set $[\![P]\!]\mathsf{tt}$. This set is then normalised as explained in § 7.3.1, and the constraints are generated. Finally, given the costs to channels and a label of interest, the tool computes all the cheapest assignments satisfying the problem Π defined in § 7.4.2.

As for the engine, we have implemented the translation from processes to backward constraints, and the loop discussed above on top of Z3 (Java API). The tool resorts on Z3 for numerical cost sets, optimising the sum of the costs. As for symbolic and non-linearly-ordered cost sets, a finite lattice can be fed into the tool, and the least upper bound is used as monoid operator. Costs can be specified in two ways: numeric costs can be directly fed to the tool, while before specifying symbolic costs the finite lattice $(\mathcal{K}, \sqsubseteq)$ has to be loaded. In order to specify a lattice, one as to declare $\top$ and $\bot$, and then operator $\oplus$ as a list of entries $x \oplus y = z$. The names of the elements of the lattice and the partial order $\sqsubseteq$ are automatically inferred from the graph of $\oplus$.

7.6 Concluding Remarks

Awareness of vulnerabilities and of their exploitation costs is an essential ingredient for developing robust systems while facing unavoidable budget considerations. In the design of software, physical, and cyber-physical systems, security is often perceived as a qualitative need, but can only be attained quantitatively. Especially when physical components are involved, it is simply impossible to predict and confront any possible attack. Even if it were possible, it would be unrealistic to have an unlimited budget to implement security mechanisms.

This perspective has found rich soil in the study of countermeasures to DoS, as witnessed by the discussion in § 3.1.3, and therefore needed to be integrated in our investigation of unavailability. The protection analysis we presented has both the merit of automatically inferring the attacks to which a system is subject, among those accountable for in the framework, and to estimate the effort required of a lucky attacker for bypassing the protection mechanisms in place.

Hence, the approach enables to identify potential weak paths and compare desired with actual protection. In terms of resilience to unavailability, the analysis facilitate estimating the cost an adversary incur to carry out a successful DoS attack.

Moreover, the framework allows reasoning with symbolic and non-linearly ordered cost structures, as it is often more natural and informative to describe the relationships between different protection mechanisms instead of assigning them absolute numbers. We showed how the analysis applies to real scenarios giving meaningful insights on the problem of password recovery. Finally, the SMT-based optimisation technique proposed for computing the analysis is exploitable in all the contexts where propositional models have to be ranked. Therefore, on the practical side, a stand-alone version of the solution engine would be highly desirable, even if recently a number of solvers are being developed.

Finally, it would be interesting to elaborate on the notion of cost set, passing from the current view, essentially static, to a dynamic setting in which costs to channels are dependent on each other and on the order in which channels are guessed. Moreover, as already highlighted by Meadows [Mea01], the monoid of the cost set needs not be commutative, as the order in which costs are paid might influence their combination. It would be interesting to investigate mechanisms for re-determining costs dynamically, as a process is evaluated.

Our work is inspired by a successful strand of literature in protocol verification, where a protocol is translated into a set of first-order Horn clauses and resolution-based theorem proving is used to establish security properties [Pau98, Wei99], and by the flow logic approach to static analysis. In particular, the translation from processes to propositional formulae is inspired by ProVerif [Bla09] translation of protocols into first-order Horn clauses, but can be more formally understood as a flow logic where the carrier logic is not the usual Alternation-free Least Fixed Point Logic, since it cannot express optimisation problems.

In order to formalise the "need for protection" of a location we resort to confidentiality lattices, that are widely used for describing levels of security in access control policies. An excellent introductory reference is [Amo94, Chs. 6,7].

Co-processes have been in studied, e.g., in [CDM14] in connection with problem of lock-freedom and progress.

As for the solution technique we exploit, different approaches have been presented to solve optimisation problems via SMT. In particular, Nieuwenhuis and Oliveras [NO06] proposed to modify the DPLL(T) procedure inherent to SMT solvers so as to look for optimal assignments, while Cimatti et al. [CFG+10] developed the search for an optimal assignments on top of an SMT solver, as we do in § 7.4.2. Nonetheless, both these works focus on numeric weights, which in our settings are represented with linearly ordered cost structures. Our more general notion of weight is modelled after Meadows's cost sets, formalised as monoids in [Mea01]. At the time of submitting this dissertation, a new version

of Microsoft Z3 for optimisation purposes is available [BAD14].

Finally, another perspective on the technical developments underpinning the analysis points to computing *prime implicants* of a given formula [DDMA12], where in our case primality is sought with respect to the cost set.

Chapter 8

Generating Attack Trees

The protection analysis of Ch. 7, moving from the urgency of quantify DoS-resilience, defines a general framework for inferring attacks. Building on the same intuition, we shall develop a systematic way to characterise graphically how an attack is achieved, in terms of composition of sub-goals and information required to attain them. We shall use *attack trees* as the graphical formalism for representing attacks. Like those of the previous chapter, the following developments overcome the boundaries of the problem of unavailability, and are strongly motivated by the necessity of devising effective way of communicating security information.

Attack trees are a widely-used graphical formalism for representing threat scenarios, as they appeal both to scientists, for it is possible to assign them a formal semantics, and to practitioners, for they convey their message in a concise and intuitive way. In an attack tree, the root represents a target goal, while the leaves contain basic attacks whose further refinement is impossible or can be neglected. Internal nodes show how the sub-trees have to be combined in order to achieve the overall attack, and to this purpose propositional conjunction and disjunction are usually adopted as combinators. On top of this basic model, a number of extensions and applications of attack trees have been proposed, demonstrating how flexible and effective a tool they are in practice. Figure 8.1 displays a simplistic attack tree, where the overall goal of entering a bank vault is obtained by either bribing a guard or by stealing the combination and neutralising the alarm.

Historically, attack trees are produced manually by teams of experts, known as Red Teams. When considering complex and sizeable systems, however, the

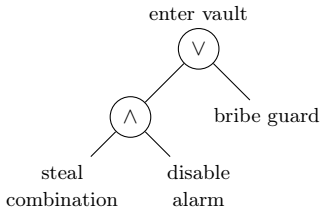


Figure 8.1: How to enter a bank vault, for dummies.

manual construction of attack trees becomes error-prone and necessarily not exhaustive. Automated techniques are therefore needed to infer complete and succinct attack trees from formal specifications. Existing approaches, surveyed in § 8.7, all suffer from focusing on computer networks, and thus suggest specification languages tailored to this domain. Moreover, the model-checking techniques that have been proposed recently for generating attack trees lead to an exponential explosion of the state space, limiting the applicability of automated search procedures.

In order to overcome these drawbacks, we develop a static analysis approach where attack trees are automatically inferred from process algebraic specifications in a syntax-directed fashion. The advantage of resorting to process calculi is many-fold. First, a process algebraic specification requires focusing on the structural and functional definition of a system, and from this deriving the threat scenario automatically, rather than thinking of it from the start, as it seems necessary with existing approaches. Second, the affinity of process calculi with programming languages established their usefulness in the formal design of complex systems, that are described in terms of interacting components. In turn, formal specifications enable the automated verification of behavioural properties at design time, hence before the actual system is produced, matching an ever-growing need for deploying software that lives up to given requirements in terms of security, safety, and performance. Finally, and most importantly, process calculi have proven useful formal languages for describing software systems, organisations, and physical infrastructure in a uniform manner.

As in the previous chapter, we define attacks as set of channels and we assume that a system P is deployed in a hostile environment, simulated by an adversary process Q running in parallel with P, resorting to the same attacker model.

Technically, we start from the translation into propositional formulae of § 7.3.1 and we obtain a tree by means of a backward reasoning procedure (§ 8.3) that connects required channels to other channels leading to them. Again, we temper the essentially qualitative nature of the translation with a quantification of the *cost* Q incurs to learn a channel, i.e., the effort related to obtaining some information. Therefore, after having computed an attack tree T_l for l, our ques-

tion will be what are the attacks of minimal cost among those described by T_l. In this sense, our attack trees encompass both the qualitative and quantitative analyses of Ch. 7.

On the complexity side, being syntax-driven, static analysis often enjoys better scalability than model-checking approaches. Even though the theoretical complexity of our analysis is still exponential in the worst case, such a price depends on the shape of the process under study, and is not incurred systematically as when a model checker generates the state space.

In general, the interplay between static analysis and model checking had a key role in advancing the community's knowledge on the foundations of formal verification, and therefore it is interesting to complement the existing studies on the attack tree generation problem with static analysis tools.

A Java implementation of the framework, briefly described in § 8.5.2, is available, which takes as input a process in the Value-Passing Quality Calculus and a location of interest, and displays graphically the corresponding attack tree. Levering the cost map, the tool also computes the cheapest sets of atomic attacks leading to l, exploiting the procedure introduced in Ch. 7. The usefulness of the framework is demonstrated on the study of the *NemID* system, a national-scale authentication system used in Denmark to provide secure Internet communication between citizens and public institutions as well as private companies.

This work is mainly based on [VNR14a] and on work currently under submission.

8.1 The NemID System

NemID (literally: EasyID)[1] is an asymmetric cryptography-based log-in solution for on-line banking and public on-line services in Denmark, used by virtually every person who resides in the country. Most service providers rely on a Java applet log-in application distributed by a national contractor, and through which their customers can be authenticated. For technological and historical reasons, the applet allows proving one's identity with various sets of credentials. In particular, private citizens can log-in with their social security number, password, and a one-time password, or by exhibiting an X.509-based certificate. Moreover, on mobile platforms that do not support Java, a user is authenticated through a classic id-password scheme.

The system is modelled in the Value-Passing Quality Calculus as follows:

$$NemID \triangleq (\nu \mathsf{login}) \ldots (\nu \mathsf{access})(!Login \mid !Applet \mid !Mobile)$$

[1] https://www.nemid.nu/dk-en/

$$
\begin{aligned}
&\mathit{Applet} \triangleq \\
&\quad {}^{1}\&_{\exists}(\mathsf{cert}?x_{\mathsf{cert}}, \&_{\forall}(\mathsf{id}?x_{\mathsf{id}}, \mathsf{pwd}?x_{\mathsf{pwd}}, \mathsf{otp}?x_{\mathsf{otp}})).\\
&\quad {}^{2}\mathsf{case}\ x_{\mathsf{cert}}\ \mathsf{of}\ \mathsf{some}(y_{\mathsf{cert}}) : {}^{3}\mathsf{login!ok}\ \mathsf{else}\\
&\qquad {}^{4}\mathsf{case}\ x_{\mathsf{id}}\ \mathsf{of}\ \mathsf{some}(y_{\mathsf{id}}) :\\
&\qquad\quad {}^{5}\mathsf{case}\ x_{\mathsf{pwd}}\ \mathsf{of}\ \mathsf{some}(y_{\mathsf{pwd}}) :\\
&\qquad\qquad {}^{6}\mathsf{case}\ x_{\mathsf{otp}}\ \mathsf{of}\ \mathsf{some}(y_{\mathsf{otp}}) : {}^{7}\mathsf{login!ok}\ \mathsf{else}\ 0\\
&\qquad\quad \mathsf{else}\ 0\\
&\qquad \mathsf{else}\ 0
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Mobile} \triangleq\ & {}^{8}\&_{\forall}(\mathsf{id}?x'_{\mathsf{id}}, \mathsf{pin}?x_{\mathsf{pin}}).\\
& {}^{9}\mathsf{case}\ x'_{\mathsf{id}}\ \mathsf{of}\ \mathsf{some}(y'_{\mathsf{id}}) :\\
& \quad {}^{10}\mathsf{case}\ x_{\mathsf{pin}}\ \mathsf{of}\ \mathsf{some}(y_{\mathsf{pin}}) : {}^{11}\mathsf{login!ok}\ \mathsf{else}\ 0\\
& \mathsf{else}\ 0
\end{aligned}
$$

$$
\mathit{Login} \triangleq\ {}^{12}\mathsf{login}?x.{}^{13}\mathsf{access!ok}
$$

The system consists of three processes running in parallel an unbounded number of times. For the sake of brevity, we have omitted to list all the restrictions in front of the parallel components, that involve all the names occurring in the three processes.[2]

Process *Login* is in charge of granting access to the system: whenever a user is authenticated via the applet or a mobile app, an output on channel login is triggered, which is received at label 12 leading to the output at label 13, which simulates a successful authentication.

Process *Applet* models the applet-based login solution, where login is granted (simulated by the outputs at label 3 and 7) whenever the user exhibits a valid certificate or the required triple of credentials. The quality binder at label 1 implements such a security check: in order to pass the binder, either ($\exists$) a certificate has to be provided, simulated by the first sub-binder, or three inputs have to be received ($\forall$), mimicking the knowledge of an id (id), a password (pwd), and a one-time password (otp).

Finally, process *Mobile* describes the intended behaviour of the mobile login solution developed by some authorities (e.g., banks, public electronic mail system), where an id and a password or pin have to be provided upon login.

In the following, we shall see how an attack tree is inferred automatically given a process P and a label l, according to the following plan:

1. P is translated into a set containing propositional formulae stating the dependency between the knowledge of channels and expressing the relationship between such knowledge and the reachability of locations (§ 8.2);

[2] Though this formulation is slightly imprecise, as for instance the mobile app is not in the scope of the password used by Java applet, we have already seen in Ch. 7 how restrictions are ignored by the analysis.

2. backward chaining the formulae representing P, a formula $[\![l]\!]$ is synthesised, stating what channels have to be in the knowledge of Q so as to drive P to l; a parse tree of such formula is an attack tree showing the combinations of channels that allow reaching l (§ 8.3);

3. given a map from channels to costs, we compute the set of minimal-cost attacks that allow reaching l among those described by the tree.

Each step will be demonstrated on the *NemID* example introduced above.

8.2 From Processes to Propositional Formulae

In the following, we shall rely on the translation $[\![P]\!]\mathsf{tt}$ devised in Table 7.3 but embrace a slightly different interpretation so as to allow explicitly generating trees. In particular, we replace bi-implications in $P^l_{\Leftrightarrow}$ with implications, obtaining a set of constraints denoted by $P^l_{\Rightarrow}$, and we do not introduce guessing literals. For the sake of simplicity, we again assume that same names and variables are defined only once.

Translating NemID. The translation of the NemID system of § 8.1 returns the following flow constraints, augmented as explained above. For the sake of simplicity we omit the occurrences of tt as a conjunct in all left-hand sides.

$$\left.\begin{array}{l}
\overline{1} \\
\overline{\mathsf{cert}} \Leftrightarrow \overline{x}_{\mathsf{cert}} \\
\overline{\mathsf{id}} \Leftrightarrow \overline{x}_{\mathsf{id}} \\
\overline{\mathsf{pwd}} \Leftrightarrow \overline{x}_{\mathsf{pwd}} \\
\overline{\mathsf{otp}} \Leftrightarrow \overline{x}_{\mathsf{otp}} \\
\underbrace{\overline{\mathsf{cert}} \vee (\overline{\mathsf{id}} \wedge \overline{\mathsf{pwd}} \wedge \overline{\mathsf{otp}})}_{\varphi} \Leftrightarrow \overline{2} \\
\varphi \wedge \overline{x}_{\mathsf{cert}} \Leftrightarrow \overline{3} \\
\varphi \wedge (\neg \overline{x}_{\mathsf{cert}}) \Leftrightarrow \overline{4} \\
\varphi \wedge (\neg \overline{x}_{\mathsf{cert}}) \wedge \overline{x}_{\mathsf{id}} \Leftrightarrow \overline{5} \\
\varphi \wedge (\neg \overline{x}_{\mathsf{cert}}) \wedge \overline{x}_{\mathsf{id}} \wedge \overline{x}_{\mathsf{pwd}} \wedge \overline{x}_{\mathsf{otp}} \Leftrightarrow \overline{7}
\end{array}\right\} \text{from } [\![\mathit{Applet}]\!]\mathsf{tt}$$

$$\left.\begin{array}{l}
\overline{8} \\
\overline{\mathsf{id}} \Leftrightarrow \overline{x'}_{\mathsf{id}} \\
\overline{\mathsf{pin}} \Leftrightarrow \overline{x}_{\mathsf{pin}} \\
\overline{\mathsf{id}} \wedge \overline{\mathsf{pin}} \Leftrightarrow \overline{9} \\
\overline{\mathsf{id}} \wedge \overline{\mathsf{pin}} \wedge \overline{x'}_{\mathsf{id}} \Leftrightarrow \overline{10} \\
\overline{\mathsf{id}} \wedge \overline{\mathsf{pin}} \wedge \overline{x'}_{\mathsf{id}} \wedge \overline{x}_{\mathsf{pin}} \Leftrightarrow \overline{11}
\end{array}\right\} \text{from } [\![\mathit{Mobile}]\!]\mathsf{tt}$$

$$\left.\begin{array}{l}
\overline{12} \\
\overline{\mathsf{login}} \Leftrightarrow \overline{x} \\
\overline{\mathsf{login}} \Leftrightarrow \overline{13} \\
g_{\mathsf{access}} \vee \overline{\mathsf{login}} \Leftrightarrow \overline{\mathsf{access}}
\end{array}\right\} \text{from } [\![\mathit{Login}]\!]\mathsf{tt}$$

$$\Big(g_{\mathsf{login}} \vee \underbrace{(\varphi \wedge \overline{x}_{\mathsf{cert}})}_{[\![Applet]\!]\mathsf{tt}} \vee \underbrace{((\varphi \wedge (\neg\overline{x}_{\mathsf{cert}}) \wedge \overline{x}_{\mathsf{id}} \wedge \overline{x}_{\mathsf{pwd}} \wedge \overline{x}_{\mathsf{otp}}))}_{[\![Applet]\!]\mathsf{tt}} \vee \underbrace{\big((\overline{\mathsf{id}} \wedge \overline{\mathsf{pin}} \wedge \overline{x'}_{\mathsf{id}} \wedge \overline{x}_{\mathsf{pin}})\big)}_{[\![Mobile]\!]\mathsf{tt}} \Big)$$

$$\Leftrightarrow \ \overline{\mathsf{login}}$$

where the last formula combines the constraints that show the various ways to trigger an output on channel login. Notice how each formula models the checks on a given path: for being granted access, i.e., reaching label 13, a communicating process has to know channel login, as specified by the constraints derived from $[\![Login]\!]\mathsf{tt}$. In turn, the last formula describes what is needed in order to get hold of login, giving rise to a backward search procedure formalised in § 8.3.

Modularity and refinement. It is worthwhile highlighting the modularity of process algebraic specification, which results in a high degree of flexibility when analysing complex systems. In the example above, for instance, while the Java applet is developed by a national contractor, and hence is common to all service providers, each company offers its own mobile app. Assume that a new way to access the system were offered by a bank, which would authenticate a user via a phone number:

$$\begin{aligned} &\mathit{Phone} \triangleq \\ &\ {}^{14}\mathsf{phone}?x_{\mathsf{ph}}.{}^{15}\mathsf{case}\ x_{\mathsf{ph}}\ \mathsf{of}\ \mathsf{some}(y_{\mathsf{ph}}) \colon {}^{16}\mathsf{login}!\mathsf{ok}\ \mathsf{else}\ 0 \\ \\ &\mathit{NemID}' \triangleq (\nu\mathsf{login}) \ldots (\nu\mathsf{phone}) \\ &\qquad\qquad (!\mathit{Login} \mid !\mathit{Applet} \mid !\mathit{Mobile} \mid !\mathit{Phone}) \end{aligned}$$

Then we have $[\![\mathit{NemID}']\!]\mathsf{tt} = [\![\mathit{NemID}]\!]\mathsf{tt} \cup [\![\mathit{Phone}]\!]\mathsf{tt}$, that is, the translation of a new top-parallel process is independent from the formulae that have already been generated. Obviously, due care has to be paid to names, e.g., name login in *Phone* has to be the same used in *NemID*. However, while restrictions play a crucial role in the semantics, they are simply ignored by the translation.

Besides being flexible with respect to the analysis of new components, the translation suitably integrates in a refinement cycle, where we start from a coarse abstraction of the system and then progressively refine those components that are revealed as candidates for being attacked, by replacing the corresponding set of formulae with a finer one. The constraint on names translates to a constraint on the interface of the component: if process A is replaced by process B, then B must be activated by the same inputs that activate A, and vice-versa, it must produce the same outputs towards the external environment that A is producing.

8.3 Synthesising Attack Trees

Once a process has been translated into propositional formulae, it is possible to build an attack tree for each program point automatically, showing what information has to be obtained and how it has to be combined in order to attain the desired goal. In order to obtain attack trees as commonly defined in the literature, in the following we assume that all the quality guards q in the process under study are linear.

Given a process P and a label l occurring in P, we generate a formula $[\![l]\!]$ representing the attack "l is reached" by backward chaining the formulae in $P^l_{\Rightarrow}$ so as to derive $\bar{l}$. It is central to observe that the procedure re-establish the original system of bi-implications thus guaranteeing the correctness of the analysis in terms of compatibility with the developments of Ch. 7.

Before explaining the algorithm, it is worthwhile discussing the nature of the backward chaining-like procedure defined in the following. Standard backward chaining [RN09, Ch. 7] combines Horn clauses so as to check whether a given goal follows from the knowledge base. Instead, we are in fact trying to derive all the knowledge bases that allow inferring the goal given the inference rules $P^l_{\Rightarrow}$, which are not strict Horn clauses as they can contain more than one positive literal. The backward-chaining point of view stresses the relationship of our problem to the quest for implicants of $\bar{l}$, as we have already observed.

8.3.1 From formulae to attack trees

The rules for generating $[\![l]\!]$ are displayed in Table 8.1. For our formulae are propositional, there is no unification other than syntactical identity of literals involved in the procedure. Notice that the algorithm only applies valid inference rules.

Rule (Sel) selects the antecedent of the formula leading to the goal $\bar{l}$: since there is a unique such rule, in order to derive $\bar{l}$ we have to derive the antecedent φ of $\varphi \Rightarrow \bar{l}$. Observe that we are not interested in deriving $\bar{l}$ in any other way: for $\bar{l}$ is derived assuming φ, the original bi-implication format $\varphi \Leftrightarrow \bar{l}$ is re-established.

Rule (Pone-c) encodes either a tautology (if $\bar{c}$ has to be inferred then $\bar{c}$ is in the knowledge base) or applications of modus ponens ($\bar{c}$ is derived assuming φ, thanks to $\varphi \Rightarrow \bar{c}$): the whole rule is an instance of disjunction introduction. Observe that putting c itself in the knowledge base is equivalent to considering literals g but keep the number of symbol – hence the tree – smaller.

This is the point where our algorithm differs from plain backward chaining: since we are building the knowledge bases that allow inferring $\bar{l}$, whenever we encounter a literal $\bar{c}$ we need to account for all the ways of deriving $\bar{c}$, namely by placing $\bar{c}$ itself in the knowledge base or by satisfying a rule whose consequent is $\bar{c}$. Similarly, rule (Pone-x) encodes an application of modus ponens, taking advantage of the uniqueness of $\varphi \Rightarrow \overline{x}$ (cf. Lemma D.1.1).

Rules (Tolle-) collect applications of modus tollens, in the classic backward fashion (i.e., when considering the derivation from the leaves to the root such steps would encode that modus). Rules (DM-) encode De Morgan's laws. Finally, rules (Comp-) simply state the compositionality of the procedure.

It is worthwhile observing that in classic backward chaining loops are avoided by checking whether a new sub-goal (i.e., a literal to be derived) is already on the goal stack (i.e., is currently being derived). Component $\mathcal{D}$ in Table 8.1 is in charge of keeping track of the current goals, but this is done on a local basis as opposed to the traditional global stack, that would result if $\mathcal{D}$ were treated as a global variable. As shown below, in our setting the global stopping criterion would lead to unsound results. Moreover, observe that using the local environment $\mathcal{D}$ we lose the linear complexity in $|P^{l}_{\Rightarrow}|$ typical of backward chaining, and incur an exponential complexity in the worst case. Nonetheless, observe that this theoretical bound is not incurred systematically.

Notice that we do not need to keep track of literals $\overline{x}$ in $\mathcal{D}$, as we cannot meet with a cycle for a variable cannot be used prior to its definition (in virtue of Lemma D.1.1).

Finally, observe that a parse tree T_l of $[\![l]\!]$ is an attack tree, showing how l can be attained by combining the knowledge of given channels. The internal nodes of the tree contain a Boolean operator in $\{\wedge, \vee\}$, while the leaves contain literals representing the knowledge of channels. As De Morgan's laws are used to push negations to literals of $[\![l]\!]$, negation can only occurs in the leaves of T_l. This approach is in line with the literature, where propositional formulae are interpreted as denotations of attack trees (e.g., see [RSF+09]). In the following, we shall manipulate attack trees always at their denotation level.

For the sake of discussion, it is worthwhile noticing that the procedure for generating $[\![l]\!]$ can be used to generate a tree explicitly during the computation, or even an And-Or graph [RN09, Ch. 4]. It is unclear to us, however, whether the more compact graph representation would be simpler to understand.

Finally, observe that $[\![l]\!]$ only contains literals $\overline{c}$ corresponding to channels, that is, the backward chaining-like procedure described above and formalised in Table 8.1 eliminates all the literals $\overline{x}$. Therefore, the reachability of l is only expressed in terms of knowledge of channels. This result is formalised in Lemma D.1.3, and will be levered in § 8.4 in order to guarantee that a map from channels to cost suffices to quantify an attack.

The global stopping criterion. Let us briefly discuss why the global stopping criterion is unsound for the procedure of Table 8.1. Consider the following set of formulae:

$$\overline{a} \Rightarrow \overline{b} \qquad \overline{b} \Rightarrow \overline{a} \qquad \overline{a} \wedge \overline{b} \Rightarrow \overline{7}$$

which stems from a conveniently simplified translation of the process

$$P \triangleq {}^{1}a?x_a.{}^{2}b!b \,|\, {}^{3}b?x_b.{}^{4}a!a \,|\, {}^{5}a?x'_a.{}^{6}b?x'_b.{}^{7}c!c$$

Table 8.1: Synthesising the propositional formula $[\![l]\!]$ for the attack tree T_l.

$$[\![l]\!] = [\![\varphi]\!]\emptyset \qquad \text{where } (\varphi \Rightarrow \bar{l}) \in P^l_{\Rightarrow} \qquad \text{(Sel)}$$

$$[\![\bar{c}]\!]\mathcal{D} = \bar{c} \vee \begin{cases} [\![\varphi]\!](\mathcal{D} \cup \{\bar{c}\}) & \text{if } \bar{c} \notin \mathcal{D}, \text{ where } (\varphi \Rightarrow \bar{c}) \in P^l_{\Rightarrow} \\ \mathsf{ff} & \text{otherwise} \end{cases} \qquad \text{(Pone-c)}$$

$$[\![\neg\bar{c}]\!]\mathcal{D} = \begin{cases} [\![\neg\varphi]\!](\mathcal{D} \cup \{\neg\bar{c}\}) & \text{if } \neg\bar{c} \notin \mathcal{D}, \text{ where } (\varphi \Rightarrow \bar{c}) \in P^l_{\Rightarrow} \\ \mathsf{tt} & \text{otherwise} \end{cases} \qquad \text{(Tolle-c)}$$

$$[\![\bar{x}]\!]\mathcal{D} = [\![\varphi]\!]\mathcal{D} \qquad \text{where } (\varphi \Rightarrow \bar{x}) \in P^l_{\Rightarrow} \qquad \text{(Pone-x)}$$

$$[\![\neg\bar{x}]\!]\mathcal{D} = [\![\neg\varphi]\!]\mathcal{D} \qquad \text{where } (\varphi \Rightarrow \bar{x}) \in P^l_{\Rightarrow} \qquad \text{(Tolle-x)}$$

$$[\![\neg(\varphi_1 \wedge \cdots \wedge \varphi_n)]\!]\mathcal{D} = [\![\neg\varphi_1]\!]\mathcal{D} \vee \cdots \vee [\![\neg\varphi_n]\!]\mathcal{D} \qquad \text{(DM-1)}$$
$$[\![\neg(\varphi_1 \vee \cdots \vee \varphi_n)]\!]\mathcal{D} = [\![\neg\varphi_1]\!]\mathcal{D} \wedge \cdots \wedge [\![\neg\varphi_n]\!]\mathcal{D} \qquad \text{(DM-2)}$$

$$[\![\varphi_1 \wedge \cdots \wedge \varphi_n]\!]\mathcal{D} = [\![\varphi_1]\!]\mathcal{D} \wedge \cdots \wedge [\![\varphi_n]\!]\mathcal{D} \qquad \text{(Comp-1)}$$
$$[\![\varphi_1 \vee \cdots \vee \varphi_n]\!]\mathcal{D} = [\![\varphi_1]\!]\mathcal{D} \vee \cdots \vee [\![\varphi_n]\!]\mathcal{D} \qquad \text{(Comp-2)}$$

$$[\![\mathsf{tt}]\!]\mathcal{D} = \mathsf{tt} \qquad [\![\mathsf{ff}]\!]\mathcal{D} = \mathsf{ff}$$

The generation of $[\![7]\!]$ unfolds as follows:

$$[\![7]\!] = [\![\bar{a} \wedge \bar{b}]\!]\emptyset = [\![\bar{a}]\!]\emptyset \wedge [\![\bar{b}]\!]\emptyset$$

where, in particular, it is

$$[\![\bar{a}]\!]\emptyset = \bar{a} \vee [\![\bar{b}]\!]\{\bar{a}\} = \bar{a} \vee \bar{b} \vee [\![\bar{a}]\!]\{\bar{a}, \bar{b}\} = \bar{a} \vee \bar{b} \vee \mathsf{ff} = \bar{a} \vee \bar{b}$$
$$[\![\bar{b}]\!]\emptyset = \bar{b} \vee [\![\bar{a}]\!]\{\bar{b}\} = \bar{b} \vee \bar{a} \vee [\![\bar{b}]\!]\{\bar{b}, \bar{a}\} = \bar{b} \vee \bar{a} \vee \mathsf{ff} = \bar{b} \vee \bar{a}$$

leading to $[\![7]\!] = \bar{a} \vee \bar{b}$, which is consistent with the reachability of label 7 in P.

Assume now to carry out the generation of $[\![7]\!]$ applying a global stopping criterion, that is, to keep track of derived goals in a global environment, initially empty. We would obtain:

$$[\![\bar{a}]\!]\emptyset = \bar{a} \vee [\![\bar{b}]\!]\{\bar{a}\} = \bar{a} \vee \bar{b} \vee [\![\bar{a}]\!]\{\bar{a}, \bar{b}\} = \bar{a} \vee \bar{b} \vee \mathsf{ff} = \bar{a} \vee \bar{b}$$

at this point, however, the environment contains $\bar{a}, \bar{b}$, and thus the generation of $[\![\bar{b}]\!]$ leads to $\bar{b}$, resulting in $[\![7]\!] = (\bar{a} \vee \bar{b}) \wedge \bar{b}$, which is not satisfied by the model where only $\bar{a}$ is tt, and thus is wrong. Analogously, we would obtain a wrong result if we chose to unfold $[\![\bar{b}]\!]$ before $[\![\bar{a}]\!]$.

8.3.2 Attacking NemID

Consider the process *NemID* discussed in § 8.1 and its translation $NemID_{\Rightarrow}$. Label 13 marks the point where a user is authenticated into the system, and

therefore is a location of interest for our analysis. Figure 8.2(a) shows the attack tree T_{13}, as generated by our implementation, presented in § 8.5.2. The backward-chaining procedure takes about 1 second on an ordinary laptop. The denotation of T_{13} is given by the following formula:

$$\begin{aligned}[\![13]\!] \;=\; & \overline{\mathsf{login}} \vee \\ & \big((\overline{\mathsf{cert}} \vee (\overline{\mathsf{id}} \wedge \overline{\mathsf{pwd}} \wedge \overline{\mathsf{otp}})) \wedge \overline{\mathsf{cert}}\big) \vee \\ & \big((\overline{\mathsf{cert}} \vee (\overline{\mathsf{id}} \wedge \overline{\mathsf{pwd}} \wedge \overline{\mathsf{otp}})) \wedge (\neg\overline{\mathsf{cert}}) \wedge \overline{\mathsf{id}} \wedge \overline{\mathsf{pwd}} \wedge \overline{\mathsf{otp}}\big) \vee \\ & \big(\overline{\mathsf{id}} \wedge \overline{\mathsf{pin}}\big)\end{aligned}$$

As a matter of fact, the algorithm tends to generate simple but redundant formulae, that can be simplified automatically, e.g., via a reduction to a normal form. The following formula, for instance, is equivalent to $[\![13]\!]$ but highlights more clearly the ways in which an attack can be carried out:

$$\overline{\mathsf{login}} \;\vee\; (\overline{\mathsf{id}} \wedge \overline{\mathsf{pin}}) \;\vee\; (\overline{\mathsf{id}} \wedge \overline{\mathsf{pwd}} \wedge \overline{\mathsf{otp}}) \;\vee\; \overline{\mathsf{cert}}$$

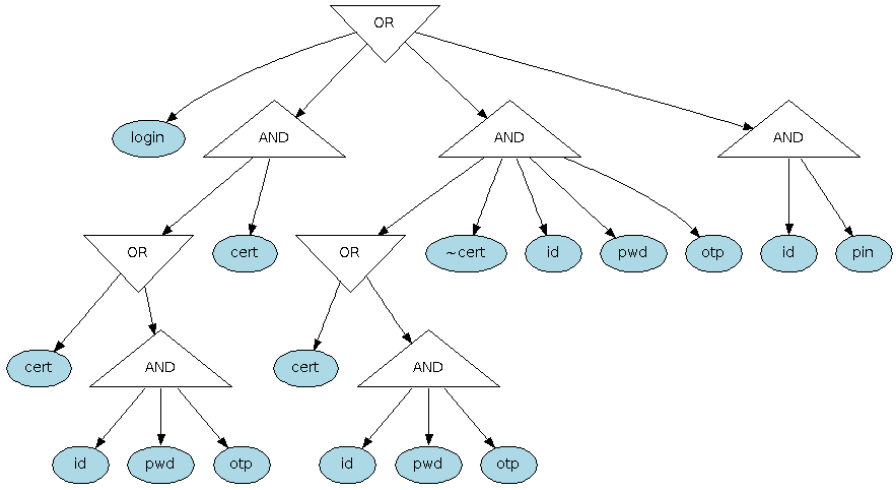
Observe that the formula above is in Disjunctive Normal Form (DNF). Such normal form has the merit of providing an immediate intuition of the alternative conditions that lead to attain the program point under study, as displayed in Fig. 8.2(b). However, the conversion to DNF may cause an exponential blow-up in the number of literals, and compact translations require to introduce fresh atoms, garbling the relation between the tree and the original system. Therefore, we did not implement such conversion in the tool.

Finally, notice that the disjunct $\overline{\mathsf{login}}$ encodes the possibility of obtaining a login token in any other way not foreseen in the system, and thus accounts for all the attacks not explicitly related to the shape of our formalisation. Such component can be disregarded by assigning it the maximum possible cost, as we shall see in the next section.

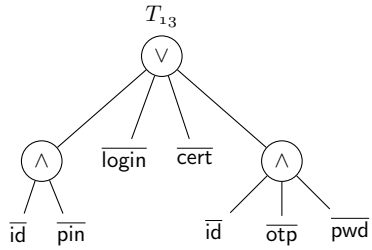
8.4 Assessing Attack Trees

A number of quantitative problems have been defined on attack trees and their extensions [KMS12]. Once a tree is characterised as a logical formula, however, a great many of them can be reduced to the problem of computing a satisfying assignment that is minimal (or, dually, maximal) with respect to a given notion of cost, hence the SMT-based solution technique presented in § 7.4.2 can be exploited.

As in Ch. 7, we resort to a map from channels to a cost set. For $[\![\ell]\!]$ only contains literals related to channels, this is enough to quantify an attack trees. Finally, observe that the quest for assignments of *minimal* cost integrates with the backward-chaining procedure of § 8.3, that avoids deriving a literal twice in the same sub-tree. In this sense, our analysis is qualitative with respect to the number of attempts are made to guess a channel: whenever the adversary



(a) T_{13} as displayed by the Quality Tree Generator, presented in § 8.5.2.



(b) The simplified DNF attack tree.

Figure 8.2: The attack tree T_{13} of the running example.

decides to incur the corresponding cost, a channel is disclosed. Afain, this is in line with the protection analysis.

There are several techniques for quantifying the cost of guessing secret information. *Quantification of information leakage* [BLMW13] is an information theory-based approach for estimating the information an adversary gains about a given secret s by observing the behaviour of a program parametrised on s. If s is quantified in bits, then the corresponding information leaked by the program is quantified as the number of bits learnt by the adversary by observing one execution of the system. For instance, consider a test program T parametrised on a secret password. T inputs a string and answers whether or not the password is matched. Under the assumptions that the adversary knows the program and the length of the secret (no security-by-obscurity), we can estimate the knowledge

gained by the adversary after one guessing attempt.

We leverage QUAIL [BLT13], a freely-available tool for quantifying information leakage, for determining costs to channels. Denoted $\lambda_T(s)$ the leakage of T on a secret s, we quantify the strength of a channel c of n bits as

$$\mathsf{cost}(c) = \frac{n}{\lambda_T(c)}$$

where we assume the security offered by c to be uniformly distributed over the n bits. In this settings we are thus working in the cost monoid $(\mathbb{Q}, +)$.

In our running example, the secrets to be guessed are pwd, otp, cert, pin, while we assume that id is known to the attacker and thus has cost 0 (in particular, in the *NemID* system is not difficult to retrieve such id, corresponding to the social security number of an individual). Moreover, we know that pwd contains between 6 and 40 alphanumeric symbols and it is not case sensitive: assuming an average length of 10 symbols, given that there are 36 such symbols, we need 5.17 bits to represent each symbol, for a total length of 52 bits. Analogously, we determine the length of otp as 20 bits, while the length of the pin depends on the service provider: in case of a major bank it is just 14 bits. As for the certificate, the authority is following NIST recommendations, using 2048-bit RSA keys for the time being, and for the sake of simplicity we assume that guessing an RSA key cannot be faster than guessing each of the bits individually. Finally, we disregard $\overline{\mathsf{login}}$ by assigning it the least upper bound of the costs of all the other channels. Exploiting QUAIL and the formula defined above, we obtain the following cost map:

$$\begin{array}{ll} \mathsf{cost}(\mathsf{pwd}) = 4.4 \times 10^{15} & \mathsf{cost}(\mathsf{pin}) = 1.5 \times 10^{4} \\ \mathsf{cost}(\mathsf{otp}) = 10^{6} & \mathsf{cost}(\mathsf{cert}) = 3.4 \times 10^{616} \end{array}$$

Fed to the SMT-based optimisation engine, the problem is found to be satisfiable with cheapest attack $\{\mathsf{id}, \mathsf{pin}\}$, whose cost is 1.5×10^4 bits, meaning that the most practicable way to break the authentication protocol is attacking the mobile app, as long as we believe that our cost map is sensible.

We have shown one elegant way of quantifying the cost of guessing a channel in the monoid $(\mathbb{Q}, +)$, but any cost map suitable to a specific application can be used. As a matter of fact, however, a cryptographer would deem the assumptions above unrealistic for estimating the strength of RSA keys, hence again a symbolic approach might prove more advisable.

8.5 Implementation

8.5.1 Comparing the protection analysis with attack trees

Let us briefly comment on upon the relationship between the protection analysis of Ch. 7 and the generation of attack trees. An attack tree displays all the attacks

leading to the given target, and thus corresponds to the qualitative analysis of § 7.3. However, the SMT-based optimisation procedures of § 7.4.2 can be used to compute minimal models of $[\![l]\!]$, hence implementing the quantitative analysis of § 7.4. In this sense, the developments on attack trees encompass fully the protection analysis.

More in detail, since the backward-chaining procedure on $P^l_{\Rightarrow}$ re-establishes bi-implications and only applies valid inference rules, $[\![l]\!]$ and $P^l_{\Leftrightarrow}$ are equisatisfiable. They are not equivalent, i.e., in general their models do not coincide, as $[\![l]\!]$ only contains channel literals, but they contain the same *attacks*, in terms of sets of channels. Hence, we can solve the quantitative version of the protection analysis in either way:

- generate $P^l_{\Leftrightarrow}$, compute the models bearing minimal attacks, and extract the corresponding attacks; or,
- generate $P^l_{\Rightarrow}$, derive $[\![l]\!]$, and compute its minimal models,

finally relating the result to the security lattice by means of the function level.

It is worthwhile observing that while the procedure for generating trees is exponential in the worst case, the size of $[\![l]\!]$ is much smaller than the size of $P^l_{\Leftrightarrow}$, and therefore it is not necessarily the case that the overall running time would increase when undertaking the tree generation. Though our example set is not extensive enough for supporting any final claim, still it is interesting to comment briefly how the analyses on $P^l_{\Leftrightarrow}$ and on $[\![l]\!]$ behave in terms of running time.

Consider the NemID system. The translation to $P^l_{\Leftrightarrow}$ takes about one fourth of the time the computation of $[\![l]\!]$ takes. Solving the optimisation problem on $P^l_{\Leftrightarrow}$ takes about 1.25 the time it takes on $[\![l]\!]$. Nonetheless, the second step is much more demanding in terms of performance, so that on average the two approaches take the same amount of time. The same applies to the login system of § 7.2. Increasing the size of the process under study it seems that the generation of attack trees, while exponential in general, tends to outperform the overall analysis on $P^l_{\Leftrightarrow}$.

It is worthwhile noticing that comparing the two approaches reduces to establishing whether it is faster to find models of $P^l_{\Leftrightarrow}$ or $[\![l]\!]$, for the translation time is negligible as the size of processes increases. Even limiting to the core propositional structure of the problem, there is no conclusive answer to the question, as we have mentioned in § 2.3.

8.5.2 The Quality Tree Generator

A proof-of-concept implementation of the framework has been developed in Java and is available at

http://www.imm.dtu.dk/~rvig/quality-trees.html

together with the code for the *NemID* example described in the text.

The Quality Tree Generator extends the Quality Tool of Ch. 7 by implementing the backward-chaining procedure. The tool takes as input an ASCII representation of a Value-Passing Quality Calculus process P and generates the set $P^{l}_{\Rightarrow}$. Moreover, given a label l occurring in P, the tool generates the formula $[\![l]\!]$, and given the cost to channels computes the cheapest assignments to $[\![l]\!]$.

As for the engine, we have implemented the backward-chaining procedure of § 8.3, defining our own simple infrastructure for propositional logic, as available libraries tend to avoid the explicit representation of implications, that is instead handy in our case during the backward-chaining computation. Once the backward-chaining procedure is executed, and thus $[\![l]\!]$ has been derived, the tool can graphically represent the corresponding tree T_l, thanks to an encoding in DOT[3] and using ZGRViewer[4] for displaying it.

All these components are glued together thanks to a simple graphical interface.

8.6 First-Order Attack Trees

We present in this section an extension to the framework whose detailed development deserves to be deepened in future work. The ideas discussed in the following have not been implemented in the tool of § 8.5.2.

The notion of knowledge needed to perform an attack adopted so far shifts the semantics load on the concept of secure channel. Besides its simplicity, this abstraction proves useful to model a great many different domains and lead to a sensible notion of attack tree. Nevertheless, it seems interesting to explore less abstract scenarios, where messages exchanged over channels do enjoy a structure and their content is exploitable in the continuation. There is a substantial corpus of literature on how to extend a process calculus to handle reasoning on terms (e.g., via equational theories or pattern matching, cf. [VNR13]), but at the semantic heart of such calculi lies the capability of testing if what is received matches what was expected.

In order to fully encompass the original Quality Calculus we should introduce both testing capabilities and structured messages. We limit here to show how to deal with the first extension, as it has a wider impact on the technical developments. As a matter of fact, distinguishing between a term t and an expression $\mathsf{some}(t)$ we are already dealing with a (very simple) signature, and this gives the necessary insight onto our idea.

The syntax of the Value-Passing Quality Calculus, introduced in § 7.1, is enhanced as follows. First of all, we allow now input and output channels to range over terms t, writing $t?x$ and $t_1!t_2$. In particular, t can be a variable y, realising name-passing. Second, we update the case clause as ${}^{l}\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(t)$:

[3] http://www.graphviz.org/

[4] http://zvtm.sourceforge.net/zgrviewer.html

P_1 else P_2, allowing to check the data payload (if any) of an input variable x. The semantics of the calculus is modified accordingly:

$$
\begin{aligned}
&{}^{l}\mathsf{case}\ \mathsf{some}(c)\ \mathsf{of}\ \mathsf{some}(c)\colon P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_1\\
&{}^{l}\mathsf{case}\ \mathsf{some}(c)\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_1[c/y]\\
&{}^{l}\mathsf{case}\ \mathsf{some}(c)\ \mathsf{of}\ \mathsf{some}(c')\colon P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_2\ \mathsf{if}\ c \neq c'\\
&{}^{l}\mathsf{case}\ \mathsf{none}\ \mathsf{of}\ \mathsf{some}(c)\colon P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_2\\
&{}^{l}\mathsf{case}\ \mathsf{none}\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_2
\end{aligned}
$$

The translation from processes to formulae of § 8.2 is lifted from propositional to first-order logic, so as to account for the richer expressiveness of the case clause:

$$
\begin{aligned}
[\![{}^{l}\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(t)\colon P_1\ \mathsf{else}\ P_2]\!]\varphi \quad = \quad & [\![P_1]\!](\varphi \wedge \exists \mathsf{fv}(t).(x = \mathsf{some}(t)) \cup\\
& [\![P_2]\!](\varphi \wedge \neg(\exists \mathsf{fv}(t).(x = \mathsf{some}(t))) \cup\\
& \{\varphi \Rightarrow \bar{l}\}
\end{aligned}
$$

where $\mathsf{some}(\cdot)$ is a unary predicate, $\mathsf{fv}(t)$ denotes the variables free in t, and we write x instead of $\overline{x}$ for now x ranges over a set of optional data. Similarly, the translation of binders has now to record the term to which an input variable is bound when the corresponding binder is satisfied:

$$\mathsf{th}(\varphi, t?x) = \{\exists y.(\varphi \wedge t \Rightarrow (x = \mathsf{some}(y))\}$$

where t ranges over a set of data (the translation of output has to be updated similarly).

Finally, for building the tree some unification is needed in the backward-chaining search of § 8.3:

$$
\begin{aligned}
&[\![\exists \mathsf{fv}(t)(x = \mathsf{some}(t))]\!]\mathcal{D} = [\![\varphi\sigma]\!]\mathcal{D}\\
&\qquad \mathsf{where}\ \ (\exists \mathsf{fv}(t')(\varphi \Rightarrow (x = \mathsf{some}(t')))) \in P^{l}_{\Rightarrow} \ \wedge\ \exists \sigma.t = t'\sigma
\end{aligned}
$$

$$
\begin{aligned}
&[\![\neg\exists \mathsf{fv}(t)(x = \mathsf{some}(t))]\!]\mathcal{D} = [\![\neg\varphi\sigma]\!]\mathcal{D}\\
&\qquad \mathsf{where}\ \ (\exists \mathsf{fv}(t')(\varphi \Rightarrow (x = \mathsf{some}(t')))) \in P^{l}_{\Rightarrow} \ \wedge\ \exists \sigma.t = t'\sigma
\end{aligned}
$$

where σ is a most general unifier.

We have thus shown how to lift all levels of the framework to name-passing calculi with full testing capabilities. From a high-level perspective, the extension allows inspecting how security checks are performed, while the basic developments consider checks as atomic entities, distinguishing between them through the cost map. Hence, we can think of the extension as lifting the protection analysis from being a counter-part to the robustness analysis to mimicking the more sophisticated availability analysis.

Though such an extension may sound interesting to the scientist, it is unclear to us whether the more detailed "first-order" trees would benefit their intended

users, the main risk consisting in that additional information would decrease readability drastically. In addition to this, whenever a finer-grained investigation is needed, we could take advantage of the modularity of the propositional framework, as discussed in § 8.2.

Finally, in order to carry the extension to the Quality Tree Generator of § 8.5.2, the main obstacle would be to introduce unification of terms in the backward-chaining procedure.

8.7 Concluding Remarks

The increasing complexity of IT systems demands for a formal investigation of their security properties, able to quantify the threats to which they are subject and to treat cyber and physical features in a uniform manner. Attack trees have proven a useful tool to study threat scenarios and convey them in an intuitive way, but any manual construction is doomed to be incomplete whenever the size of the tree exceeds a few hundred nodes.

In order to tackle this problem, we have presented a novel method for the automated generation of attack trees. In particular, our technique improves on the existing literature by resorting to a process-algebraic specification of the system. This choice allows to model a great many scenarios, beyond the standard network security domain, and enables designing syntax-directed static analyses, avoiding the systematic state space explosion suffered by model checking algorithms, even if retaining an exponential worst-case complexity. Moreover, process calculi have proven useful notations for the formal design of complex systems, embracing the need for analysing vulnerabilities at design time, and thus before the actual system is produced.

As far as this dissertation is concerned, this and the previous chapter show how Quality Calculi can be used to express highly-branching systems in a concise manner. This is a feature that stems from the approach to availability that informs the calculi but at the same time overcomes it, just as the developments in these chapters apply but are not limited to DoS.

The feasibility of the approach is witnessed by a freely-available implementation, and has been demonstrated on the study of a real system used for authentication purposes on a national scale.

As future work, besides consolidating the proof-of-concept implementation, it would be worth investigating in details the intuitions discussed in § 8.6 to encompass the full calculus. Finally, a direct performance comparison with existing tools based on model checking would be interesting, even though the usefulness of our approach partly lies in the capability of dealing with scenarios not encodable in those tools.

Attack trees and the like. Graphical representations of security threats are often used to convey complex information in an intuitive way. Formalisation of such graphical objects are referred to chiefly as *attack graphs* [PS98, JSW02, SW04, MBZ+06] and *attack trees* [Sch99, SHJ+02, MO06, RSF+09, JW10]. In this work we prefer the phrase "attack trees", but our procedure can be adapted to generate attack graphs.

While different authors have different views on the information that should decorate such objects, instrumental to the analysis that the tree or the graph is supporting, all definitions share the ultimate objective of showing how atomic attacks (i.e., the leaves) can be combined to attain a target goal (i.e., the root). This perspective is enhanced in the seminal work of Schneier [Sch99], that found a great many extensions and applications. In particular, Mauw and Oostdijk [MO06] lay down formal foundations for attack trees, while Kordy et al. [KMRS10] and Roy et al. [RKT12] suggest ways to unify attacks and countermeasures in a single view. Even though Schneier's work is mostly credited for having introduced attack trees, and it had certainly a crucial role in making attack trees mainstream in computer security, the origin of this formalism can be traced back to fault trees, expert systems (e.g., Kuang [Bal87]), and privilege graphs [DDK96].

Automated generation of attack trees. As for the automated generation of attack graphs, the literature is skewed towards the investigation of network-related vulnerabilities: available tools expect as input rich models, including information such as the topology of the network and the set of atomic attacks to be considered. The backward search techniques of Phillips and Swiler [PS98] and Sheyner et al. [SHJ+02] have proven useful to cope with the explosion of the state space due to such expressive models. However, the search has to be carried out on a state space that is exponential in the number of system variables, whose construction is the real bottle-neck of these approaches, and the result graph tends to be large even if compact BDD-based representations are used, as argued in [AWK02]. In particular, in [SHJ+02] a model checking-based approach is developed, where attack graphs are characterised as counter-examples to safety properties; a detailed example is discussed in [SW04]. Similarly to Phillips and Swiler, we adopt an attacker-centric perspective, which cannot simulate benign system events such as the failure of a component, as in [SHJ+02]. Directly addressing the exponential blow-up of [SHJ+02], Ammann et al. [AWK02] propose a polynomial algorithm, but the drop in complexity relies on the assumption of monotonicity of the attacker actions and on the absence of negation. On the same line, Ou et al. [OBM06] present an algorithm which is quadratic in the number of machines in the network under study.

Analysing attack trees. As for the analyses developed on top of attack trees, we present a reachability analysis which computes the cheapest sets of

atomic attacks that allow attaining a location of interest in the system, as it is standard in the attack tree literature. This approach seamlessly encompasses the probabilistic analysis of [SHJ+02, SW04] (costs to atomic attacks would represent their likelihood and the objective function would compute the overall probability) and offers a uniform framework to address other quantitative questions [BDP07, KMS12]. The NP-completeness of out SMT-based approach is in line with the complexity of the minimisation analysis of [SHJ+02, SW04]. It is worthwhile noticing that the correctness of the analysis with respect to the semantics corresponds to the *exhaustiveness* of attack trees as defined in [SHJ+02]: $[\![l]\!]$ covers all possible attacks leading to l.

Finally, Mehta et al. [MBZ+06] present a technique for ranking sub-graphs so as to draw attention to the most promising security flaws. Whilst we do not directly tackle this issue, for condensing an entire tree into a formula we gain in performance but we lose the original structure, a post-processing step could be undertaken to compute the value of the internal nodes (sub-formulae).

Chapter 9

Conclusion

> There are more things in heaven and earth, Horatio, / Than are dreamt of in your philosophy.
>
> Hamlet I:5, 167–8

In this dissertation we have explored how robustness against DoS can be enforced by means of a principled design process supported by static analysis techniques, and we have devised process-algebraic languages that facilitate specifying such analyses in a natural manner, thanks to the promotion of availability concerns as first-class objects of the discourse domain. The thesis we initially claimed was that

language-based technologies offer a unifying approach to deal with the consequences of DoS, by means of a framework for facilitating the development of programs that follow a planned behaviour when expected information is unavailable. The modelling language can be supplemented by formal analyses enforcing such robust code to be produced.

We shall now summarise the contributions of this dissertation, so as to facilitate eliciting how our developments support, and perhaps overcome, the claim above. Finally, we shall survey briefly how this work is placed in the wider landscape of formal approaches to system development, thereby highlighting promising directions for future investigation, as opposed to the technical treatment of future work presented at the end of each chapter.

9.1 Contribution

Let us briefly review the main contributions of our work on availability and arrange them orthogonally with respect to the exposition in chapters.

Coping with the consequences of unavailability. Our starting point was the observation that DoS attacks are increasingly damaging and frequent, despite a plethora of clever techniques for countering them. As a consequence, a rigorous approach is needed to cope with the effects of DoS. Besides patching the vulnerability to successful attacks and thus complementing existing approaches, addressing the consequences of unavailability encompasses all potential sources of DoS, whether they be cyber or physical, inadvertent or malicious.

Availability of data can be expressed by formal languages. The main effect of DoS in distributed systems is the unavailability of data to some components. Adopting into a process-algebraic world the well-established notion of option data type, the Quality Calculus addresses the distinction between available and unavailable data. Moreover, by means of input contracts we have further refined the source of unavailability in lacking of expected communication and mismatch between expected and received data, giving formal dignity to an intuition already advanced in early literature. In turn, capturing availability allows to reason about unavailable data and act accordingly, enforcing the most suitable behaviour given the actual information and resorting to default data when information gaps have to be filled. When it is possible to provide each component with a plan for every possible situation, then full resilience to DoS is achieved and its typical domino effect is banished.

Static analysis can pinpoint unavailability threats. Static analysis is the tool by means of which we can ensure that the most is gotten out of the expressiveness of the language. The suite of analyses we have devised addresses the various unavailability-related phenomena we can account for in a number of Quality Calculi, and it does so in an efficient way, thanks to the implementation in terms of satisfiability problems and relying on the performance of modern solvers.

Qualitative and quantitative perspectives can be reconciled. Quality Calculi allow to present graceful degradation and cost-based considerations in a common framework. First, we devised a language for enforcing the former; then, we showed how the latter can be integrated seamlessly, inducing a natural evolution in the tools that complement the language. Ideal robustness against DoS cannot be achieved without combining existing quantitative solutions and the Quality Calculus way: we should exploit the former for avoiding unavailability conditions, and rely on the latter for facing actual unavailability of some

components, which the cost-based techniques cannot rule out completely. As a result, the essentially qualitative demand for availability and the inevitable quantitative guarantees that practical solutions offer can be reconciled in a coherent language-based framework.

9.2 Future Directions

The reader is referred to the concluding remarks that follow each chapter for a discussion of the technical developments that seem to deserve further exploration. We shall adopt now a higher-level perspective and peep out through the fence into which we restricted our own investigation in Ch. 2. Reflecting on what we do not do can prove useful to highlight some neighbouring topics from which our discourse would benefit.

Attempting a negative definition in a rich discipline such as computer science is an overwhelming task, inevitably bound to left some readers dissatisfied. Nevertheless, it is worthwhile mentioning some research lines concerned with questions directly arising from the overall approach illustrated in Ch. 2.

There are two situations in which the exploitation of process calculi for formal verification *typically* takes place. Sometimes we want to know whether an existing system enjoys given properties, and to this end a model of the system is produced and analysed. This is mainly the case of legacy systems that no one would ever dare re-engineer and re-deploy, but which we are required to investigate to some extent. Another (perhaps utopian) situation is the development of a new system from scratch, where we have the possibility to start from a formal model and only after having analysed it thoroughly produce the real system "in its own image".

It is evident how our framework captures only a portion of the development cycle, and in particular lacks a connection with the real system in both the use cases depicted above. In other words, we start from a process-algebraic model P, we abstract it into a simpler model P' that enjoys a precise relationship with P, we analyse P' and lift the verification results to P. Now, what about the relationship between P and the original system? Nothing can be formally stated about its properties. To quote Dijkstra [DDH72, § I.5]:

> *If one proves the correctness of a program assuming an idealised, perfect world, one should not be amazed if something goes wrong when this ideal program gets executed by an "imperfect" implementation.*

What is worse, we do not even know whether P is an idealised model of reality or just a wrong one. Whilst it seems virtually impossible to establish such connection in case of existing systems, *refinement* techniques cope with the challenge of deriving an actual implementation from a formal model in a semi-automated manner (cf. [Wir71] for a seminal work on program development by

refinement).

As for the usefulness of defining programming abstractions that account for real world behaviours, another issue we do no tackle is the porting of such abstractions to real programming languages. This is for example the case of the Java implementation [BDP02] of KLAIM [BBD+03]. Other significant experiences are those of LOTOS and CADP (see [INR]) and JSCL [FGS06].

All these themes would be certainly worth exploring, and we shall consider examining them in depth in future work.

Finally, we do not claim to have devised "the ultimate process calculus". Instead, we studied one paradigmatic and foundational aspect of communication in distributed systems that has received little attention from the formal methods community so far. In order to better focus on this single feature, we deliberately chose to disregard a great many interesting traits tackled by other domain-specific calculi and by unifying frameworks. A good many of these features are worth studying within Quality Calculi.

With regards to this, it is worthwhile mentioning that a handful of Quality Calculi has been developed that are not covered in this dissertation. Probabilistic considerations about the trustworthiness of input are at the heart of the relational analysis devised in [RN13a], leading to a quantitative understanding of the *quality* of the various program points. The interplay between the essentially *safety*-oriented mind-set of the calculus and *security* requirements of real systems is investigated in [RN13b]. Real-world scenarios also motivate [WNR14], where unavailability of components is studied with respect to the continuous evolution of the physical world. The stochastic and true-concurrent behaviour of communicating distributed components is the topic of [ZNR14], where the semantic models induced by various probabilistic distributions are characterised.

Appendix A

Proofs for Ch. 4

This appendix contains the proof of correctness of the robustness analysis presented in § 4.6 with respect to the explicit substitution semantics of § 4.5, and the equivalence of the latter and the reduction semantics discussed in § 4.2.

A.1 Correctness of the Robustness Analysis

Let us formalise some notation introduced in § 4.6.1.

Definition A.1 ($\bar{\cdot}$ notation) Let ρ be a substitution. Then, $\overline{\rho}$ is a substitution from variables $x \in \mathcal{X}$ to Boolean values, defined as follows:

$$\mathsf{dom}(\overline{\rho}) = \{x \in \mathcal{X} \mid x \in \mathsf{dom}(\rho)\} \qquad (\overline{\rho}x) = \begin{cases} \mathsf{tt} & \text{if } (\rho x) = \mathsf{some}(c) \\ \mathsf{ff} & \text{otherwise} \end{cases}$$

Moreover, since $\mathsf{dom}(\overline{\rho}) \subseteq \mathcal{X}$, the following fact holds immediately.

Fact A.1.1 *Let ρ_1, ρ_2 be substitutions, such that $\mathsf{dom}(\rho_2) \cap \mathcal{X} = \emptyset$. Then, $\overline{\rho_2 \circ \rho_1} = \overline{\rho_1 \circ \rho_2} = \overline{\rho_1}$.*

Finally, let us recall the definition of propositional satisfiability, which is essential to the correctness statement.

Definition A.2 (Propositional satisfiability $\models$) Let ρ be a substitution and φ a propositional formula, such that the variables occurring in φ

are a subset of $\mathsf{dom}(\rho)$. We say that $\overline{\rho}$ *satisfies* or *models* φ, denoted $\overline{\rho} \models \varphi$, if φ evaluates to tt under $\overline{\rho}$, and we write $\overline{\rho} \not\models \varphi$ otherwise:

$$\begin{array}{llll}
\overline{\rho} \models \mathsf{tt} & \text{for all } \overline{\rho} & \overline{\rho} \models \varphi_1 \wedge \varphi_2 & \text{if } \ \overline{\rho} \models \varphi_1 \text{ and } \overline{\rho} \models \varphi_2 \\
\overline{\rho} \not\models \mathsf{ff} & \text{for all } \overline{\rho} & \overline{\rho} \models \varphi_1 \vee \varphi_2 & \text{if } \ \overline{\rho} \models \varphi_1 \text{ or } \overline{\rho} \models \varphi_2 \\
\overline{\rho} \models x & \text{if } (\overline{\rho}x) = \mathsf{tt} & \overline{\rho} \models \neg\varphi & \text{if } \ \overline{\rho} \not\models \varphi \\
\overline{\rho} \not\models x & \text{if } (\overline{\rho}x) = \mathsf{ff} &&
\end{array}$$

Before proving Lemmata 4.6.1,4.6.2 we shall establish some auxiliary results for accommodating the relationship between substitutions and formulae generated for expressions and binders, and for dealing with good processes.

Lemma A.1.1

$$\begin{array}{ll}
(1) & \forall e \forall \rho \left((\exists c.(\rho e) = \mathsf{some}(c)) \ \wedge \ \vdash e \triangleright \varphi_e \ \Rightarrow \ \overline{\rho} \models \varphi_e \right) \\
(2) & \forall e \forall \rho \left((\rho e) = \mathsf{none} \ \wedge \ \vdash e \triangleright \varphi_e \ \Rightarrow \ \overline{\rho} \models \neg\varphi_e \right)
\end{array}$$

Proof. By induction on the structure of expressions e. The result is immediate for the cases $\mathsf{some}(c), \mathsf{none}, x$, while the inductive step stems from the assumption of soundness and completeness of the evaluation of functions f with respect to the $\overline{\cdot}$ notation, as postulated in § 4.6.1 above:

$$[\![f]\!](\overline{o_1}, \ldots, \overline{o_n}) = \overline{o} \qquad \text{whenever} \qquad f(o_1, \ldots, o_n) \rhd o$$

□

Lemma A.1.2 *Let Θ, θ be substitutions from variables $x \in \mathcal{X}$ to constant optional data, and b be a binder. It holds:*

$$\forall \Theta \forall \theta \forall b \left((\Theta \gg b) ::_{\mathsf{tt}} \theta \ \wedge \ \vdash b \blacktriangleright \varphi_b \ \Rightarrow \ \overline{\theta} \models \varphi_b \right)$$

Proof. By induction on the structure of binder b, observing that θ is constructed according to the semantics of $::_{\mathsf{tt}}$. □

Lemma A.1.3 *For all contexts C and explicit processes S_1, S_2 it holds that*

$$\mathsf{good}(C[S_1]) \ \wedge \ \mathsf{good}(S_2) \ \Rightarrow \ \mathsf{good}(C[S_2])$$

Proof. By induction on the structure of contexts C. □

Lemma A.1.4 *For all contexts C and explicit processes S it holds that*

$$\mathsf{good}(C[S]) \ \Rightarrow \ \mathsf{good}(S)$$

PROOF. By induction on the structure of contexts C. □

LEMMA A.1.5 *For all explicit processes S_1, S_2 it holds that*

$$\mathsf{good}(S_1|S_2) \;\Leftrightarrow\; (\mathsf{good}(S_1) \wedge \mathsf{good}(S_2))$$

PROOF. ($\Rightarrow$) The result follows as a corollary of Lemma A.1.4.

($\Leftarrow$) Assume $\mathsf{good}(S_1)$ and $\mathsf{good}(S_2)$. The result follows immediately by definition of goodness, observing that $S_1|S_2$ can be written as a context of processes that are good, i.e., $C[S_1]$ and $C[S_2]$. □

We can now show Lemma 4.6.1: the structural congruence preserves goodness of explicit processes.

PROOF.[of Lemma 4.6.1] Let us first recall the statement of the lemma:

$$\forall S, S' \,.\, (\mathsf{good}(S) \wedge S \Rrightarrow S' \;\Rightarrow\; \mathsf{good}(S'))$$

The proof is organised by induction on the shape of the inference tree T for the congruence step $S \Rrightarrow S'$.

Basis. The basis of the induction consists of the case analysis of all the axiom schemas in Table 4.6.

(Ref) Assume that T consists of an application of rule (Ref), and assume $\mathsf{good}(S)$. The thesis follows immediately.

(Nil) Assume that T consists of an application of rule (Nil), and assume $\mathsf{good}(S|\{\rho\}\mathsf{0})$. The thesis $\mathsf{good}(S)$ follows directly from Lemma A.1.4.

(Nil') Assume that T consists of an application of rule (Nil'), and assume $\mathsf{good}(S)$. The thesis $\mathsf{good}(S|\{\rho\}\mathsf{0})$ follows observing that $S|\{\rho\}\mathsf{0}$ can be written as a context of either component, that $\Phi_{P_1|P_2} = \Phi_{P_1} = \Phi_{P_2}$, and by the hypothesis $\mathsf{good}(S)$.

(Com) Assume that T consists of an application of rule (Com), and assume $\mathsf{good}(S_1|S_2)$. The thesis follows directly from Lemma A.1.5.

(Ass) Assume that T consists of an application of rule (Ass), and assume $\mathsf{good}(S_1|(S_2|S_3))$. The thesis $\mathsf{good}((S_1|S_2)|S_3)$ follows observing that $\Phi_{P_1|(P_2|P_3)} = \Phi_{(P_1|P_2)|P_3}$ and by Lemma A.1.4, which ensures $\mathsf{good}(S_i)$ for $i \in [1,3]$.

(New1) Assume that T consists of an application of rule (New1), and assume $\mathsf{good}((\nu c_1)\,(\nu c_2)\,S)$. Then, by Lemma A.1.4 $\mathsf{good}(S)$ holds, and the thesis $\mathsf{good}((\nu c_2)\,(\nu c_1)\,S)$ follows directly from the definition of goodness as the restriction are absorbed in a context surrounding S (i.e., $\Phi_{(\nu c_1)\,(\nu c_2)\,P} = \Phi_{(\nu c_2)\,(\nu c_1)\,P} = \Phi_P$).

(New2) Assume that T consists of an application of rule (New2), and assume $\mathsf{good}((\nu c)\,\{\rho\}P)$ with $c \notin \mathsf{fc}(\{\rho\}P)$. Then, the thesis $\mathsf{good}(\{\rho\}P)$ follows directly from Lemma A.1.4.

(New2') Assume that T consists of an application of rule (New2'), and assume $\mathsf{good}(\{\rho\}P)$ with $c \notin \mathsf{fc}(\{\rho\}P)$. Then, the thesis $\mathsf{good}((\nu c)\,\{\rho\}P)$ follows observing that $(\nu c)\,\{\rho\}P$ can be written as a context of $\{\rho\}P$ and that $\Phi_{(\nu c)\,P} = \Phi_P$.

(New3) Assume that T consists of an application of rule (New3), and assume $\mathsf{good}((\nu c)\,(\{\rho_1\}P_1|\{\rho_2\}P_2))$. The thesis $\mathsf{good}(((\nu c)\,\{\rho_1\}P_1)|\{\rho_2\}P_2)$ follows observing that $\Phi_{((\nu c)\,P_1)|P_2} = \Phi_{(\nu c)\,(P_1|P_2)} = \Phi_{P_i}$, for $i \in [1,2]$, and that $(\nu c)\,(\{\rho_1\}P_1|\{\rho_2\}P_2)$ can be written as $C_i[\{\rho_i\}P_i]$ for proper C_1, C_2.

(New3') Analogous to the previous case hence omitted.

(S-par) Assume that T consists of an application of rule (S-par), and assume $\mathsf{good}(\{\rho\}(P_1|P_2))$, from which $\overline{\rho} \models \Phi_{(P_1|P_2)}$. Since $\Phi_{(P_1|P_2)} = \Phi_{P_i}$, for $i \in [1,2]$, and $S' = C[\{\rho\}P_i]$, the thesis $\mathsf{good}(\{\rho\}P_1|\{\rho\}P_2)$ follows.

(S-new) Assume that T consists of an application of rule (S-new), and assume $\mathsf{good}(\{\rho\}(\nu c)\,P)$, from which $\overline{\rho} \models \Phi_{(\nu c)\,P}$. Observe that $\Phi_{(\nu c)\,P} = \Phi_P$ and recall that $\overline{\rho \circ [c'/c]} = \overline{\rho}$. Then, the result follows since S' has the form $C[\{\rho\}P]$ and $\overline{\rho} \models \Phi_P$.

(S-case) Assume that T consists of an application of rule (S-case), and assume

$$\mathsf{good}(\{\rho\}\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2)$$

from which $\overline{\rho} \models \Phi_{\mathsf{case}}$. Since S is closed, in S' the expression (ρe) is either $\mathsf{some}(c)$ or none. Assume $(\rho e) = \mathsf{some}(c)$ and $\vdash e \triangleright \varphi_e$: by Lemma A.1.1 it follows $\overline{\rho} \models \varphi_e$. Moreover, observe that $\Phi_{P_1} = \Phi_{\mathsf{case}} \wedge \varphi_e$, therefore by definition of satisfaction relation it holds that $\overline{\rho} \models \Phi_{P_1}$, and thus the thesis follows since $\overline{[c/y] \circ \rho} = \overline{\rho}$.
The same reasoning applies to the case $(\rho e) = \mathsf{none}$.

(S-bin) Assume that T consists of an application of rule (S-bin), and assume $\mathsf{good}(\{\rho\}(b.P'))$, from which $\overline{\rho} \models \Phi_{b.P}$. Moreover, assume $(\Theta \gg b) ::_{\mathsf{tt}} \theta$ and $\vdash b \blacktriangleright \varphi_b$, from which $\Phi_P = \Phi_{b.P} \wedge \varphi_b$. Since the domains of ρ and θ are disjoint and $\overline{\rho} \models \Phi_{b.P}$, proving $\overline{\theta \circ \rho} \models \Phi_P$ is equivalent to proving $\overline{\theta} \models \varphi_b$, which follows from Lemma A.1.2. Finally, the result follows observing that $S' = C[(\rho b).\{\rho\}P]$.

(S-out) Assume that T consists of an application of the rule (S-out), and assume $\mathsf{good}(\{\rho\}(t_1!t_2.P))$, from which $\overline{\rho} \models \Phi_{t_1!t_2.P}$. The thesis follows observing that $\Phi_P = \Phi_{t_1!t_2.P}$, and that $S' = C[(\rho t_1)!(\rho t_2).\{\rho\}P]$.

Step. The inductive step consists of the case analysis of the composite rules in Table 4.6.

(Tra) Assume that T consists of an application of rule (Tra), where $S_1 \Rrightarrow S_3$ is derived thanks to $S_1 \Rrightarrow S_2$ and $S_2 \Rrightarrow S_3$, and assume $\mathsf{good}(S_1)$. By inductive hypothesis, we have

$$\mathsf{good}(S_i) \wedge S_i \Rrightarrow S_{i+1} \;\Rightarrow\; \mathsf{good}(S_{i+1})$$

for $i \in [1, 2]$, from which the result follows immediately.

(Cnt) Assume that T consists of an application of rule (Cnt), where $C[S_1] \Rrightarrow C[S_2]$ is derived thanks to $S_1 \Rrightarrow S_2$, and assume $\mathsf{good}(C[S_1])$. As observed in Lemma A.1.4, $\mathsf{good}(S_1)$ follows. Then, by inductive hypothesis, it holds

$$\mathsf{good}(S_1) \wedge S_1 \Rrightarrow S_2 \;\Rightarrow\; \mathsf{good}(S_2)$$

and the result follows directly by Lemma A.1.3. □

It remains to show the proof of Lemma 4.6.2. In order to ease the presentation, we first show an auxiliary lemma linking the step-wise evaluation of binders to their satisfiability properties.

Lemma A.1.6 *Let Θ, θ be substitutions from variables $x \in \mathcal{X}$ to constant optional data, and b be a binder. It holds:*

$$\forall\Theta\forall\theta\ (\vdash b \blacktriangleright \varphi_b \wedge\ (\Theta \gg b) ::_{\mathsf{tt}} \theta \quad\Rightarrow\quad \overline{\theta} \models \varphi_b) \tag{A.1}$$

$$\Downarrow$$

$$\forall\Theta\forall\theta\forall c_1\forall c_2\ (c_1!c_2 \vdash b \to b' \ \wedge \vdash b' \blacktriangleright \varphi_{b'} \wedge (\Theta \gg b') ::_{\mathsf{tt}} \theta \quad\Rightarrow\quad \overline{\theta} \models \varphi_{b'})$$

Proof. By an immediate induction on the inference $c_1!c_2 \vdash b \to b'$ it follows $b' = \Theta' \gg b$, for some substitution Θ'.
By induction on the judgement $\vdash b \blacktriangleright \varphi_b$ it follows $\vdash (\Theta \gg b) \blacktriangleright \varphi_b$, for all substitutions Θ. In particular, this is the case because the judgement produces the same formula x for the binders $t?x$, $\Theta \gg t?x$, and $[\mathsf{some}(c)/x]$, and because $[\mathsf{none}/x]$ is never produced.
Now suppose $c_1!c_2 \vdash b \to b'$, $\vdash b' \blacktriangleright \varphi_{b'}$, $(\Theta \gg b') ::_{\mathsf{tt}} \theta$, and that assumption (A.1) holds. Since $b' = \Theta' \gg b$, we can write $((\Theta \circ \Theta') \gg b) ::_{\mathsf{tt}} \theta$, for some Θ'. Moreover, by hypothesis we have $\overline{\theta} \models \varphi_b$, where $\vdash b \blacktriangleright \varphi_b$. Finally, since $\varphi_b = \varphi_{b'}$, the conclusion $\overline{\theta} \models \varphi_{b'}$ follows. □

Proof.[of Lemma 4.6.2] Let us first recall the statement of the lemma:

$$\forall S, S'.\ (\mathsf{good}(S) \wedge S \longrightarrow S' \;\Rightarrow\; \mathsf{good}(S'))$$

The proof is organised by induction on the shape of the inference tree T for the transition $S \longrightarrow S'$.

Basis. The are five possible cases to be considered, corresponding to rules (In-), (Case-), and (Rec) in Table 4.7.

(In-ff) Assume that T consists of an application of rule (In-ff), and assume $\mathsf{good}(c_1!c_2.\{\rho_1\}P_1 \mid b.\{\rho_2\}P_2)$. The following two cases are possible:

- $S = C[c_1!c_2.\{\rho_1\}P_1]$ with $\overline{\rho_1} \models \Phi_{P_1}$. Hence, for $S' = C'[\{\rho_1\}P_1]$ it follows $\mathsf{good}(S')$, since $\Phi_{P_1} = \Phi_{c_1!c_2.P_1}$.
- $S = C[b.\{\rho_2\}P_2]$ with $S' = C'[b'.\{\rho_2\}P_2]$. Then, from the hypothesis $\mathsf{good}(S)$, it follows

$$\forall\Theta\forall\theta\left((\Theta \gg b) ::_{\mathsf{tt}} \theta \Rightarrow \overline{\theta \circ \rho_2} \models \Phi_P\right)$$

and we shall show

$$\forall\Theta\forall\theta'\left((\Theta \gg b') ::_{\mathsf{tt}} \theta' \Rightarrow \overline{\theta' \circ \rho_2} \models \Phi'_P\right)$$

where, assuming $\vdash b \blacktriangleright \varphi_b$, we have $\Phi_P = \Phi'_P = (\varphi \wedge \varphi_b)$ for some φ, since we have already observed in Lemma A.1.6 that $\vdash b' \blacktriangleright \varphi_b$. Moreover, since the domain of ρ_2 is disjoint from the domains of θ and θ' (variables are bound exactly once), it must be $\overline{\rho} \models \varphi$. Finally, as $b' = (\Theta' b)$ for some Θ', from Lemma A.1.6 it follows $\overline{\theta'} \models \varphi_b$, and the thesis follows by definition of satisfaction relation.

(In-tt) Analogous to the previous case hence omitted.

(Case-tt) Assume that T consists of an application of rule (Case-tt), and assume $\mathsf{good}(\mathsf{case}\ \mathsf{some}(c)\ \mathsf{of}\ \mathsf{some}(y) : \{\rho_1\}P_1\ \mathsf{else}\ \{\rho_1\}P_2)$, from which $\overline{\rho_1} \models \Phi_{P_1}$. Then, we have $S' = C'[\{[c/y] \circ \rho_1\}P_1]$, and the thesis follows since $\overline{[c/y] \circ \rho_1} = \overline{\rho_1} \models \Phi_{P_1}$.

(Case-ff) Analogous to the previous case hence omitted.

(Rec) Assume that T consists of an application of rule (Rec), where A evolves to its definition P. Now, P must be one of the P_i's defined in P_*, thus we have $\Phi_P = \mathsf{tt}$, and the result follows trivially since $S' = C[\{[(\rho e)/x]\}P]$ and $\mathsf{id} \models \mathsf{tt}$.

Step. There are two remaining cases, corresponding to the composite rules of the semantics.

(Cng) Assume that T consists of an application of rule (Cng):

$$\frac{S_1 \Rrightarrow S_2 \quad S_2 \longrightarrow S_3 \quad S_3 \Rrightarrow S_4}{S_1 \longrightarrow S_4}$$

and assume $\mathsf{good}(S_1)$. By Lemma 4.6.1, it holds

$$\mathsf{good}(S_i) \wedge S_i \Rrightarrow S_{i+1} \;\Rightarrow\; \mathsf{good}(S_{i+1})$$

for $i \in \{1, 3\}$, and in particular we have $\mathsf{good}(S_2)$. Moreover, by inductive hypothesis we have

$$\mathsf{good}(S_2) \wedge S_2 \longrightarrow S_3 \;\Rightarrow\; \mathsf{good}(S_3)$$

from which the result $\mathsf{good}(S_4)$ follows by Lemma 4.6.1.

(Cnt) Assume that T consists of an application of rule (Cng), where $C[S_1] \longrightarrow C[S_2]$ is inferred from $S_1 \longrightarrow S_2$, and assume $\mathsf{good}(C[S_1])$. As observed in the proof of Lemma A.1.5, $\mathsf{good}(S_1)$ follows. Then, by inductive hypothesis we have $\mathsf{good}(S_2)$, and the result follows by Lemma A.1.3. □

A.2 Semantic Equivalence

It remains to be be shown the equivalence between the reduction semantics of § 4.2 and the explicit substitution semantics of § 4.5, on which the correctness of the robustness analysis has been proven.

In the following we shall thus relate explicit processes S produced by the explicit substitution semantics to processes P produced by the reduction semantics. This is obtained by applying the substitution under which an explicit process is active to the process itself, thereby obtaining a process compatible with the reduction semantics. The following definition establishes such a correspondence, where due care has to be paid to avoid accidental capture of bound names.

Definition A.3 ($\overline{\cdot}$ notation) Let S be an explicit process. Then, the *instance* P of S, denoted $P = \overline{\overline{S}}$, is structurally defined as follows:

$$\overline{\overline{\{\rho\}0}} = 0$$

$$\overline{\overline{\{\rho\}(\nu c)\,P}} = \begin{cases} (\nu c)\overline{\overline{\{\rho\}P}} & \text{if } c \notin \mathsf{fc}(\{\rho\}P) \\ (\nu c')\overline{\overline{\{(\rho \circ [c'/c])\}P}} & \text{if } c \in \mathsf{fc}(\{\rho\}P) \wedge c' \notin \mathsf{fc}(\{\rho\}P) \end{cases}$$

$$\overline{\overline{\{\rho\}\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2}} = \mathsf{case}\ (\rho e)\ \mathsf{of}\ \mathsf{some}(y)\colon \overline{\overline{\{\rho\}P_1}}\ \mathsf{else}\ \overline{\overline{\{\rho\}P_2}}$$

$$\overline{\overline{\{\rho\}b.P}} = (\rho b).\overline{\overline{\{\rho\}P}} \qquad \overline{\overline{\{\rho\}t_1!t_2.P}} = (\rho t_1)!(\rho t_2).\overline{\overline{\{\rho\}P}}$$

$$\overline{\overline{S_1|S_2}} = \overline{\overline{S_1}}|\overline{\overline{S_2}} \qquad \overline{\overline{\{\rho\}(P_1|P_2)}} = \overline{\overline{\{\rho\}P_1}}|\overline{\overline{\{\rho\}P_2}}$$

$$\overline{\overline{t_1!t_2.S}} = t_1!t_2.\overline{\overline{S}} \qquad \overline{\overline{(\nu c)\,S}} = (\nu c)\,\overline{\overline{S}}$$

$$\overline{\overline{b.S}} = b.\overline{\overline{S}} \qquad \overline{\overline{\{\rho\}A(e)}} = A((\rho e))$$

$$\overline{\overline{\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon S_1\ \mathsf{else}\ S_2}} = \mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon \overline{\overline{S_1}}\ \mathsf{else}\ \overline{\overline{S_2}}$$

Observe that we have $\overline{\overline{\{\mathsf{id}\}P_*}} = P_*$, thus relating the starting point of a semantic evaluation in the two semantics. In the following, we shall use the symbol $\xrightarrow{\equiv}$ to denote the reduction semantics and the symbol $\xrightarrow{\Rrightarrow}$ to denote the explicit substitution semantics, stressing in particular the two notions of congruence they are parametrised on.

Theorem A.2.1 ($\xrightarrow{\equiv} = \xrightarrow{\Rrightarrow}$) *For all systems*

$$\begin{array}{ll} \mathsf{define} & A_1(x_1) \triangleq P_1 \\ & \vdots \\ & A_n(x_n) \triangleq P_n \\ \mathsf{in} & \{\mathsf{id}\}P_* \end{array}$$

it holds that

$$(1)\quad \forall S\,.\,\Big(\{\mathsf{id}\}P_* \xrightarrow{\Rrightarrow}{}^{*}\ S\ \Rightarrow\ P_* \xrightarrow{\equiv}{}^{*}\ \overline{\overline{S}}\Big)$$

$$(2)\quad \forall P\,.\,\Big(P_* \xrightarrow{\equiv}{}^{*}\ P\ \Rightarrow\ \exists S\,.\,\Big(\{\mathsf{id}\}P_* \xrightarrow{\Rrightarrow}{}^{*}\ S \wedge \overline{\overline{S}} = P\Big)\Big)$$

Clause (1) in the theorem claims that the reduction semantics can simulate the explicit substitution semantics, while clause (2) states that the explicit substitution semantics can simulate the reduction semantics. The equivalence of $\xrightarrow{\equiv}$ and $\xrightarrow{\Rrightarrow}$ then follows.

For the sake of readability, we shall separate the two results in Lemmata A.2.2, A.2.5 below. Each of these lemmata relies on an auxiliary result linking the directed structural congruence $\Rrightarrow$ on explicit processes to the structural congruence $\equiv$ on processes.

Lemma A.2.1 ($\Rrightarrow \subseteq \equiv$)

$$\forall S, S'\,.\ \Big(S \Rrightarrow S'\ \Rightarrow\ \overline{\overline{S}} \equiv \overline{\overline{S'}}\Big)$$

Proof. The proof is organised by induction of the shape of the inference tree T for the congruence step $S \Rrightarrow S'$.

Basis. The basis of the induction consists of the case analysis of all the axiom schemas in Table 4.6.

(Ref) Assume that T consists of an application of rule (Ref). Then, it holds that $\overline{\overline{S}} \equiv \overline{\overline{S}}$ and the thesis follows.

(Nil) Assume that T consists of an application of rule (Nil). Then, it holds that $\overline{\overline{S|\{\rho\}\mathsf{0}}} = \overline{\overline{S}|\overline{\{\rho\}\mathsf{0}}} = \overline{\overline{S}|\mathsf{0}} \equiv \overline{\overline{S}}$, and the thesis follows.

(Nil') Analogous to the previous case hence omitted.

(Com) Assume that T consists of an application of rule (Com). Then, it holds that $\overline{\overline{S_1|S_2}} = \overline{\overline{S_1}|\overline{S_2}} \equiv \overline{\overline{S_2}|\overline{S_1}} = \overline{\overline{S_2|S_1}}$, and the thesis follows.

(Ass) Assume that T consists of an application of rule (Ass). Then, it holds that $\overline{\overline{S_1|(S_2|S_3)}} = \overline{\overline{S_1}|(\overline{\overline{S_2}|\overline{S_3}})} \equiv \overline{(\overline{\overline{S_1}|\overline{S_2}})|\overline{S_3}} = \overline{\overline{(S_1|S_2)|S_3}}$, and the thesis follows.

(New1) Assume that T consists of an application of rule (New1). Then, it holds that $\overline{\overline{(\nu c_1)\,(\nu c_2)\,S}} = (\nu c_1)\,(\nu c_2)\,\overline{\overline{S}} \equiv (\nu c_2)\,(\nu c_1)\,\overline{\overline{S}} = \overline{\overline{(\nu c_2)\,(\nu c_1)\,S}}$, and the thesis follows.

(New2) Assume that T consists of an application of rule (New2) and assume $c \notin \mathsf{fc}(\{\rho\}P)$. Then, it holds that $\overline{\overline{(\nu c)\,\{\rho\}P} = (\nu c)\,\overline{\overline{\{\rho\}P}} \equiv \overline{\overline{\{\rho\}P}}}$, and the thesis follows.

(New2') Analogous to the previous case hence omitted.

(New3) Assume that T consists of an application of rule (New3) and assume $c \notin \mathsf{fc}(\{\rho_2\}P_2)$. Then it holds that

$$
\begin{array}{lll}
\overline{\overline{((\nu c)\,\{\rho_1\}P_1)|\{\rho_2\}P_2}} & = & [\text{def. of } \bar{\cdot}] \\
\overline{\overline{((\nu c)\,\{\rho_1\}P_1)}|\overline{\{\rho_2\}P_2}} & = & [\text{def. of } \bar{\cdot}] \\
((\nu c)\,\overline{\overline{\{\rho_1\}P_1}})|\overline{\overline{\{\rho_2\}P_2}} & \equiv & [\text{rule (New3}')/\text{(New3)}\,] \\
(\nu c)\,(\overline{\overline{\{\rho_1\}P_1}}|\overline{\overline{\{\rho_2\}P_2}}) & = & [\text{def. of } \bar{\cdot}] \\
(\nu c)\,\overline{(\overline{\{\rho_1\}P_1}|\overline{\{\rho_2\}P_2})} & = & [\text{def. of } \bar{\cdot}] \\
\overline{\overline{(\nu c)\,(\{\rho_1\}P_1|\{\rho_2\}P_2)}} & &
\end{array}
$$

(New3') Analogous to the previous case hence omitted.

(S-par) Assume that T consists of an application of rule (S-par). Then, it holds that $\overline{\overline{\{\rho\}(P_1|P_2)} = \overline{\overline{\{\rho\}P_1}|\overline{\{\rho\}P_2}} = \overline{\overline{\{\rho\}P_1|\{\rho\}P_2}}}$, and thesis follows.

(S-new) Assume that T consists of an application of rule (S-new). Then, it holds that

$$
\overline{\overline{\{\rho\}((\nu c)\,P)}} = \begin{cases} (\nu c)\overline{\overline{\{\rho\}P} = \overline{\overline{(\nu c)\{\rho\}P}} & \text{if } c \notin \mathsf{fc}(\{\rho\}P) \\ \overline{(\nu c')\overline{\{(\rho \circ [c'/c])\}P} = & \text{if } c \in \mathsf{fc}(\{\rho\}P) \wedge c' \notin \mathsf{fc}(\{\rho\}P) \\ \overline{(\nu c')\{(\rho \circ [c'/c])\}P}} & \end{cases}
$$

and thesis follows since processes are equal up to α-renaming, we can safely push/pull substitutions beyond the restriction undertaking the necessary renaming.

(S-case) Assume that T consists of an application of rule (S-case). Then, it holds that $\overline{\overline{\{\rho\}\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2} = \overline{\mathsf{case}\ (\rho e)\ \mathsf{of}\ \mathsf{some}(y) : \overline{\{\rho\}P_1}\ \mathsf{else}\ \overline{\{\rho\}P_2}} = \overline{\mathsf{case}\ (\rho e)\ \mathsf{of}\ \mathsf{some}(y)\colon \{\rho\}P_1\ \mathsf{else}\ \{\rho\}P_2}}$, and thesis follows.

(S-bin) Assume that T consists of an application of rule (S-bin). Then, it holds that $\overline{\overline{\{\rho\}b.P} = (\rho b).\overline{\{\rho\}P} = \overline{(\rho b).\{\rho\}P}}$, and the thesis follows.

(S-out) Assume that T consists of an application of rule (S-out). Then, it holds that $\overline{\overline{\{\rho\}t_1!t_2.P} = (\rho t_1)!(\rho t_2).\overline{\{\rho\}P} = \overline{(\rho t_1)!(\rho t_2).\{\rho\}P}}$, and the thesis follows.

Step.

(Tra) Assume that T consists of an application of rule (Tra), where $S_1 \Rrightarrow S_3$ is inferred thanks to $S_1 \Rrightarrow S_2$, $S_2 \Rrightarrow S_3$. By inductive hypothesis, we have

$$S_i \Rrightarrow S_{i+1} \;\Rightarrow\; \overline{\overline{S_i}} \equiv \overline{\overline{S_{i+1}}}$$

for $i \in [1,2]$, from which the result follows immediately by transitivity of $\equiv$.

(Cnt) Assume that T consists of an application of rule (Cnt), where $C[S_1] \Rrightarrow C[S_2]$ is inferred thanks to $S_1 \Rrightarrow S_2$. By inductive hypothesis, it holds

$$S_1 \Rrightarrow S_2 \;\Rightarrow\; \overline{\overline{S_1}} \equiv \overline{\overline{S_2}}$$

and the result follows according to the preservation of $\equiv$ in contexts, for the application of $\overline{\overline{\cdot}}$ to contexts for explicit processes produces contexts for processes (and is deterministic). □

Lemma A.2.2 ($\xrightarrow{\Rrightarrow} \subseteq \xrightarrow{\equiv}$) *For all systems, it holds that*

$$\forall S \,.\, \left(\{\mathsf{id}\} P_* \xrightarrow{\Rrightarrow}{}^{*} S \;\Rightarrow\; P_* \xrightarrow{\equiv}{}^{*} \overline{\overline{S}} \right)$$

Proof. The proof is organised by induction on the length k of the derivation sequence $\{\mathsf{id}\}P_* \xrightarrow{\Rrightarrow}{}^{*} S$.

Basis. Assume $k = 0$. Then we have $\{\mathsf{id}\}P_* = S$, from which the thesis follows since $P_* \xrightarrow{\equiv}{}^{0} P_* = \overline{\overline{\{\mathsf{id}\}P_*}}$.

Step. Assume that the result holds for $k \leq k_0$; we shall prove it for $k_0 + 1$. The whole derivation sequence can be written as

$$\{\mathsf{id}\}P_* \xrightarrow{\Rrightarrow}{}^{k_0} S' \longrightarrow S$$

and the inductive hypothesis applies to the first k_0 steps of the derivation, leading to

$$P_* \xrightarrow{\equiv}{}^{*} \overline{\overline{S'}}$$

Therefore, it suffices to show that

$$S' \xrightarrow{\Rrightarrow} S \;\Rightarrow\; \overline{\overline{S'}} \xrightarrow{\equiv} \overline{\overline{S}}$$

which follows from Lemma A.2.3 directly. □

Lemma A.2.3 (Inductive step of Lemma A.2.2)

$$\forall S, S' \,.\, \left(S \xrightarrow{\Rightarrow} S' \;\Rightarrow\; \overline{\overline{S}} \xrightarrow{\equiv} \overline{\overline{S'}} \right)$$

Proof. The proof is organised by induction on the shape of the inference tree T for the transition $S \xrightarrow{\Rightarrow} S'$.

Basis. There are five possible cases to be considered, corresponding to rules (In-), (Case-), and (Rec) of the explicit substitution semantics of Table 4.7.

(In-ff) Assume that T consists of an application of rule (In-ff). We have $\overline{\overline{S}} = c_1!c_2.\overline{\overline{\{\rho_1\}P_1}} \mid b.\overline{\overline{\{\rho_2\}P_2}}$, $\overline{\overline{S'}} = \overline{\overline{\{\rho_1\}P_1}} \mid b'.\overline{\overline{\{\rho_2\}P_2}}$. Therefore, we can build the following inference tree in the reduction semantics:

$$\frac{t_1 \rhd c_1 \quad t_2 \rhd c_2 \quad c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{ff}} \theta}{t_1!t_2.\overline{\overline{\{\rho_1\}P_1}} \mid b.\overline{\overline{\{\rho_2\}P_2}} \xrightarrow{\equiv} \overline{\overline{\{\rho_1\}P_1}} \mid b'.\overline{\overline{\{\rho_2\}P_2}}}$$

and thesis $\overline{\overline{S}} \xrightarrow{\equiv} \overline{\overline{S'}}$ follows.

(In-tt) Analogous to the previous case, observing that $\overline{\overline{\{\theta \circ \rho_2\}P_2}} = \left(\overline{\overline{\{\rho_2\}P_2}}\right)\theta$.

(Case-tt) Assume that T consists of an application of rule (Case-tt). We have $\overline{\overline{S}} = \mathsf{case}\ \mathsf{some}(c)\ \mathsf{of}\ \mathsf{some}(y) : \overline{\overline{\{\rho_1\}P_1}}\ \mathsf{else}\ \overline{\overline{\{\rho_1\}P_2}}$, $\overline{\overline{S'}} = \overline{\overline{\{[c/y] \circ \rho_1\}P_1}}$. Therefore, we can build the following inference tree in the reduction semantics:

$$\frac{e \rhd \mathsf{some}(c)}{\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y) : \overline{\overline{\{\rho_1\}P_1}}\ \mathsf{else}\ \overline{\overline{\{\rho_1\}P_2}} \xrightarrow{\equiv} \left(\overline{\overline{\{\rho_1\}P_1}}\right)[c/y]}$$

from which the thesis $\overline{\overline{S}} \xrightarrow{\equiv} \overline{\overline{S'}}$ follows, since $\overline{\overline{\{[c/y] \circ \rho_1\}P_1}} = \left(\overline{\overline{\{\rho_1\}P_1}}\right)[c/y]$.

(Case-ff) Analogous to the previous case.

(Rec) Assume that T consists of an application of rule (Rec). We have $\overline{\overline{S}} = A((\rho e))$ and $\overline{\overline{S'}} = \overline{\overline{\{[(\rho e)/x]\}P}} = P[(\rho e)/x]$. Therefore, we can build the following inference tree in the reduction semantics:

$$A((\rho e)) \xrightarrow{\equiv} P[(\rho e)/x] \quad \text{if } A(x) \triangleq P$$

from which the thesis $\overline{\overline{S}} \xrightarrow{\equiv} \overline{\overline{S'}}$ follows.

Step. There are two remaining cases, corresponding to the composite rules of the explicit substitution semantics of Table 4.7.

(Cng) Assume that T consists of an application of rule (Cng):

$$\frac{S_1 \Rrightarrow S_2 \quad S_2 \overset{\Rrightarrow}{\longrightarrow} S_3 \quad S_3 \Rrightarrow S_4}{S_1 \overset{\Rrightarrow}{\longrightarrow} S_4}$$

with $S = S_1$ and $S' = S_4$. First of all, observe that by Lemma A.2.1 it holds

$$S_i \Rrightarrow S_{i+1} \;\Rightarrow\; \overline{\overline{S_i}} \equiv \overline{\overline{S_{i+1}}}$$

for $i \in \{1, 3\}$. Moreover, by inductive hypothesis we have

$$S_2 \overset{\Rrightarrow}{\longrightarrow} S_3 \;\Rightarrow\; \overline{\overline{S_2}} \overset{\equiv}{\longrightarrow} \overline{\overline{S_3}}$$

Hence, we can build the following inference tree in the reduction semantics:

$$\frac{\overline{\overline{S_1}} \equiv \overline{\overline{S_2}} \quad \overline{\overline{S_2}} \overset{\equiv}{\longrightarrow} \overline{\overline{S_3}} \quad \overline{\overline{S_3}} \equiv \overline{\overline{S_4}}}{\overline{\overline{S_1}} \overset{\equiv}{\longrightarrow} \overline{\overline{S_4}}}$$

and the thesis follows.

(Cnt) Assume that T consists of an application of rule (Cnt), where $S = C[S_1] \overset{\Rrightarrow}{\longrightarrow} C[S_2] = S'$ is inferred thanks to $S_1 \overset{\Rrightarrow}{\longrightarrow} S_2$. By inductive hypothesis, it holds

$$S_1 \overset{\Rrightarrow}{\longrightarrow} S_2 \;\Rightarrow\; \overline{\overline{S_1}} \overset{\equiv}{\longrightarrow} \overline{\overline{S_2}}$$

and the result follows according to the preservation of the reductions for processes in contexts

$$\frac{\overline{\overline{S_1}} \overset{\equiv}{\longrightarrow} \overline{\overline{S_2}}}{\overline{\overline{C[S_1]}} \overset{\equiv}{\longrightarrow} \overline{\overline{C[S_2]}}}$$

for the application of $\overline{\cdot}$ to contexts for explicit processes produces contexts for processes. □

We have thus established the first clause of Th. A.2.1, and we shall now show the second clause, that is, the explicit substitution semantics can simulate the reduction semantics. Let us follow the approach of the former proof and show first an auxiliary result which links the structural congruence on processes to the directed congruence on explicit processes.

Lemma A.2.4 ($\equiv \subseteq \Rrightarrow$)

$$\forall P, P' \,.\, \Big(P \equiv P' \;\Rightarrow\; \forall S \,.\, \Big(\overline{\overline{S}} = P \;\Rightarrow\; \exists S' \,.\, \Big(\overline{\overline{S'}} = P' \wedge S \Rrightarrow S'\Big)\Big)\Big)$$

Proof. The proof is organised by induction on the shape of the inference tree T for the congruence step $P \equiv P'$. For the sake of clarity, we shall consider the symmetric version of each rule in the basis of the induction.

Basis. The basis consists of the case analysis of the axiom schemas in the structural congruence of Table 4.2.

(Ref) Assume that T consists of an application of rule (Ref) and consider an explicit process S such that $\overline{\overline{S}} = P$. Then, it holds that $S \Rrightarrow S$ and the result follows (as the rule is self-symmetric, no consideration about symmetry is needed).

(Nil) Assume that T consists of an application of rule (Nil) and consider an explicit process S such that $\overline{\overline{S}} = P = P_0|\mathsf{0}$. According to the definition of $\overline{\overline{\cdot}}$, S is either

- $S_0|\{\rho\}\mathsf{0}$, with $\overline{\overline{S_0}} = P_0$, in which case we conclude observing that $S_0|\{\rho\}\mathsf{0} \Rrightarrow S_0$; or
- $\{\rho\}(R_0|\mathsf{0})$, with $\overline{\overline{\{\rho\}R_0}} = P_0$, in which case we conclude observing that $S \Rrightarrow \{\rho\}R_0|\{\rho\}\mathsf{0} \Rrightarrow \{\rho\}R_0$.

Consider now the symmetric version of the rule, according to which P is re-written into $P|\mathsf{0} = P'$, and consider an explicit process S such that $\overline{\overline{S}} = P$. The result follows observing that thanks to rule (Nil') of $\Rrightarrow$ it holds that $S \Rrightarrow S|\{\rho\}\mathsf{0}$ and $\overline{\overline{S|\{\rho\}\mathsf{0}}} = \overline{\overline{S}}|\overline{\overline{\{\rho\}\mathsf{0}}} = P|\mathsf{0}$.

(Com) Assume that T consists of an application of rule (Com) and consider an explicit process S such that $\overline{\overline{S}} = P = P_1|P_2$. Then, S is either

- $S_1|S_2$, with $\overline{\overline{S_i}} = P_i$ for $i \in [1,2]$, in which case $S \Rrightarrow S_2|S_1$ and the result follows; or
- $\{\rho\}(R_1|R_2)$, with $\overline{\overline{\{\rho\}R_i}} = P_i$ for $i \in [1,2]$, in which case $S \Rrightarrow \{\rho\}R_1|\{\rho\}R_2 \Rrightarrow \{\rho\}R_2|\{\rho\}R_1$ and the result follows.

As the rule is self-symmetric, no consideration about symmetry is needed.

(Ass) Assume that T consists of an application of rule (Ass) and consider an explicit process $S = P = P_1|(P_2|P_3)$. Then, S is either

- $S_1|(S_2|S_3)$, with $\overline{\overline{S_i}} = P_i$ for $i \in [1,3]$, in which case we conclude observing that $S \Rrightarrow (S_1|S_2)|S_3$; or
- $\{\rho\}(R_1|(R_2|R_3))$, with $\overline{\overline{\{\rho\}R_i}} = P_i$, in which case we conclude observing that $S \Rrightarrow \{\rho\}R_1|(\{\rho\}(R_2|R_3) \Rrightarrow \{\rho\}R_1|(\{\rho\}R_2|\{\rho\}R_3) \Rrightarrow (\{\rho\}R_1|\{\rho\}R_2)|\{\rho\}R_3$; or

- $\{\rho_1\}R_1|\{\rho_2\}(R_2|R_3)$, with $\overline{\overline{\{\rho_1\}R_1}} = P_1$ and $\overline{\overline{\{\rho_2\}R_i}} = P_i$ for $i \in [2,3]$, and we conclude as in the previous case; or
- $\{\rho_1\}R_1|(\{\rho_1\}R_2|\{\rho_1\}R_3)$, with $\overline{\overline{\{\rho\}R_i}} = P_i$ for $i \in [1,3]$, were we conclude applying rule (Ass) of $\Rrightarrow$ directly.

As for the symmetric version of the rule, observe that it is derivable in $\Rrightarrow$ as follows:

$$\begin{array}{ll}
(S_1 \,|\, S_2) \,|\, S_3 \Rrightarrow & \text{(Com)} \\
S_3 \,|\, (S_1 \,|\, S_2) \Rrightarrow & \text{(Ass)} \\
(S_3 \,|\, S_1) \,|\, S_2 \Rrightarrow & \text{(Com)} \\
S_2 \,|\, (S_3 \,|\, S_1) \Rrightarrow & \text{(Ass)} \\
(S_2 \,|\, S_3) \,|\, S_1 \Rrightarrow & \text{(Com)} \\
S_1 \,|\, (S_2 \,|\, S_3) &
\end{array}$$

and the same reasoning as above applies.

(New1) Assume that T consists of an application of rule (New1) and consider an explicit process S such that $\overline{\overline{S}} = P = (\nu c_1)\,(\nu c_2)\,P_0$. Then, S is either

- $(\nu c_1)\,(\nu c_2)\,S_0$, with $\overline{\overline{S_0}} = P_0$, in which case we conclude applying rule (New1) of $\Rrightarrow$; or,
- $\{\rho\}(\nu c_1)\,(\nu c_2)\,R_0$, with $\overline{\overline{\{\rho\}R_0}} = P_0$, in which case we conclude by pushing the substitution beyond the restrictions (applying rule (S-new) twice) and then using rule (New1) of $\Rrightarrow$. Observe that we are free to rename c_1, c_2 when applying (S-new), obtaining a process α-equivalent to P, hence equal to P.

As the rule is self-symmetric, no consideration about symmetry is needed.

(New2) Assume that T consists of an application of rule (New2) and consider an explicit process S such that $\overline{\overline{S}} = P = (\nu c)\,P_0$ with $c \notin \mathsf{fc}(P_0)$. Then, S is either

- $(\nu c)\,S_0$, with $\overline{\overline{S_0}} = P_0$, in which case we conclude applying rule (New2) of $\Rrightarrow$; or,
- $\{\rho\}(\nu c)\,R_0$, with $\overline{\overline{\{\rho\}R_0}} = P_0$, in which case we conclude by pushing the substitution beyond the restriction (applying rule (S-new) twice) and then using rule (New2) of $\Rrightarrow$. Observe that we are free to rename c when applying (S-new), obtaining a process α-equivalent to P, hence equal to P.

Consider now the symmetric version of the rule, according to which P is re-written into $(\nu c)\,P$ provided that $c \notin \mathsf{fc}(P)$, and consider an explicit process $S = \{\rho\}P_0$ such that $\overline{\overline{S}} = P$ and $c \notin \mathsf{fc}(P_0)$. The result follows

observing that thanks to rule (New2') of $\Rrightarrow$ it holds that $S \Rrightarrow (\nu c)\, S$, and we have $\overline{\overline{(\nu c)\, S}} = (\nu c)\, P$.

(New3) Assume that T consists of an application of rule (New3) and consider an explicit process S such that $\overline{\overline{S}} = P = (\nu c)\,(P_1|P_2)$. Then, S is either

- $(\nu c)\,(\{\rho_1\}P_1|\{\rho_2\}P_2)$, with $\overline{\overline{\{\rho_i\}R_i}} = P_i$ for $i \in [1,2]$ and $c \notin \mathsf{fc}(\overline{\overline{\{\rho\}R_2}})$, in which case we conclude by applying rule (New3) of $\Rrightarrow$ directly; or
- $\{\rho\}(\nu c)\,(R_1|R_2)$, with $\overline{\overline{\{\rho\}R_i}} = P_i$ for $i \in [1,2]$ and $c \notin \mathsf{fc}(\overline{\overline{\{\rho\}R_2}})$, in which case we first have to push the substitution beyond the restriction using rule (S-new), then carry the substitution to the parallel components by rule (S-par), and finally conclude applying rule (New3) of $\Rrightarrow$.

Consider now the symmetric version of the rule, according to which $P = ((\nu c)\, P_1)|P_2$ is re-written into $(\nu c)\,(P_1|P_2)$ provided that $c \notin \mathsf{fc}(P_2)$, and consider an explicit process S such that $\overline{\overline{S}} = P$. The reasoning proceeds as above by taking advantage of rule (New3') of $\Rrightarrow$.

Step. The basis consists of the case analysis of the composite rules in the directed congruence of Table 4.6.

(Tra) Assume that T consists of an application of rule (Tra), where $P = P_1 \equiv P_3 = P'$ is derived thanks to $P_1 \equiv P_2$ and $P_2 \equiv P_3$. Consider an explicit process S_1 such that $\overline{\overline{S_1}} = P_1$. Then, by inductive hypothesis we have

$$P_1 \equiv P_2 \wedge \overline{\overline{S_1}} = P_1 \;\Rightarrow\; \exists S_2\,.\,\left(S_1 \Rrightarrow S_2 \wedge \overline{\overline{S_2}} = P_2\right)$$

and the inductive hypothesis applies again to S_2, leading to

$$P_2 \equiv P_3 \wedge \overline{\overline{S_2}} = P_2 \;\Rightarrow\; \exists S_3\,.\,\left(S_2 \Rrightarrow S_3 \wedge \overline{\overline{S_3}} = P_3\right)$$

and we conclude $S_1 \Rrightarrow S_3$ by transitivity of the directed congruence on explicit processes.

(Cnt) Assume that T consists of an application of rule (Cnt), where $P = C[P_1] \equiv C[P_2] = P'$ is derived thanks to $P_1 \equiv P_2$. Consider an explicit process S such that $\overline{\overline{S}} = C[P_1]$. We proceed by induction on the structure of the context C.

Basis. Assume that $C = [\,]$. Then, we have $P = P_1 = \overline{\overline{S}}$ and $P' = P_2$. Hence, we conclude observing that by inductive hypothesis it holds that

$$P_1 \equiv P_2 \wedge \overline{\overline{S}} = P_1 \;\Rightarrow\; \exists S_{P_2}\,.\,\left(S \Rrightarrow S_{P_2} \wedge \overline{\overline{S_{P_2}}} = P_2\right)$$

Step. There are two cases to consider.

- Assume that $C = (\nu c)\, C_0[\,]$. Then, we have $P = (\nu c)\, C_0[P_1] = \overline{\overline{S}}$ and $P' = (\nu c)\, C_0[P_2]$. Now, according to the definition of $\overline{\cdot}$, S is either $\{\rho\}(\nu c)\, C_0^S[P_1^S]$ with $\overline{\overline{\{\rho\}C_0^S[P_1^S]}} = C_0[P_1]$ or $(\nu c)\, S_0$ with $\overline{\overline{S_0}} = C_0[P_1]$. Moreover, observe that the former reduces to the latter by a congruence step, and thus we shall focus on $S = (\nu c)\, S_0$. Observe that the inductive hypothesis applies to the sub-process S_0 of S, leading to

  $$C_0[P_1] \equiv C_0[P_2] \wedge \overline{\overline{S_0}} = C_0[P_1] \;\Rightarrow\; \exists S_0' \,.\, \left(\overline{\overline{S_0'}} = C_0[P_2] \;\wedge\; S_0 \Rrightarrow S_0'\right)$$

  and we conclude observing that the congruence on explicit processes is preserved in contexts, and thus we can build the following inference tree:

  $$\frac{S_0 \Rrightarrow S_0'}{S = (\nu c)\, S_0 \Rrightarrow (\nu c)\, S_0' = S'}$$

  where $\overline{\overline{S'}} = P' = (\nu c)\, C_0[P_2]$ holds by definition of $\overline{\cdot}$, for we have $\overline{\overline{S_0'}} = C_0[P_2]$.
- The reasoning for the case $C = C_0[\,]|Q$ (and for the symmetric right-parallel context) proceeds analogously. □

Lemma A.2.5 ($\xrightarrow{\equiv} \subseteq \xrightarrow{\Rrightarrow}$) *For all systems, it holds that*

$$\forall P \,.\, \left(P_* \xrightarrow{\equiv}{}^* P \;\Rightarrow\; \exists S \,.\, \left(\{\mathsf{id}\}P_* \xrightarrow{\Rrightarrow}{}^* S \;\wedge\; \overline{\overline{S}} = P\right)\right)$$

Proof. The proof is organised by induction on the length k of the derivation sequence $P_* \xrightarrow{\equiv}{}^* P$.

Basis. Assume $k = 0$. Then $P = P_*$, and similarly in no step $\{\mathsf{id}\}P_*$ evolves to itself in the explicit substitution semantics, yielding $S = \{\mathsf{id}\}P_*$, and we conclude observing that $P = P_* = \overline{\overline{\{\mathsf{id}\}P_*}} = \overline{\overline{S}}$.

Step. Assume that the result holds for $k \leq k_0$; we shall prove it for $k_0 + 1$. Then, the whole derivation sequence can be written as

$$P_* \xrightarrow{\equiv}{}^{k_0} P' \xrightarrow{\equiv} P$$

and the inductive hypothesis applies to the first k_0 steps of the derivation, leading to

$$\exists S' \,.\, \left(\{\mathsf{id}\}P_* \xrightarrow{\Rrightarrow}{}^* S' \;\wedge\; \overline{\overline{S'}} = P'\right)$$

Therefore, it suffices to show that

$$\overline{\overline{S'}} \xrightarrow{\equiv} P \;\Rightarrow \exists S.\, \Big(S' \xrightarrow{\Rightarrow} S \wedge \overline{\overline{S}} = P\Big)$$

which follows from Lemma A.2.6 directly. □

Lemma A.2.6 (Inductive step of Lemma A.2.5)

$$\forall P, P'.\, \Big(P \xrightarrow{\equiv} P' \;\Rightarrow\; \forall S.\, \Big(\overline{\overline{S}} = P \;\Rightarrow\; \exists S'.\, \Big(\overline{\overline{S'}} = P' \wedge S \xrightarrow{\Rightarrow} S'\Big)\Big)\Big)$$

Proof. The proof is organised by induction on the shape of the inference tree T for the semantic step $P \xrightarrow{\equiv} P'$.

Basis. There are five possible cases to be considered, corresponding to rules (In-), (Case-), and (Rec) of the reduction semantics of Table 4.4.

(In-ff) Assume that T consists of an application of rule (In-ff), where $P = t_1!t_2.P_1|b.P_2$ and $P' = P_1|b'.P_2$. Consider an explicit process S such that $\overline{\overline{S}} = P$. By definition of $\bar{\cdot}$, it is either

- $S = \{\rho\}(R_1|R_2)$, where $\overline{\overline{\{\rho\}R_1}} = c_1!c_2.P_1$ and $\overline{\overline{\{\rho\}R_2}} = b.P_2$; or,
- $S = \{\rho_1\}R_1|\{\rho_2\}R_2$, where $\overline{\overline{\{\rho_1\}R_1}} = c_1!c_2.P_1$ and $\overline{\overline{\{\rho_2\}R_2}} = b.P_2$; or,
- $S = c_1!c_2.\{\rho_1\}R_1|b.\{\rho_2\}R_2$, where $\overline{\overline{\{\rho\}R_1}} = P_1$ and $\overline{\overline{\{\rho\}R_2}} = P_2$.

However, since the first two explicit processes reduce to the latter applying the directed congruence, let us focus on $S = c_1!c_2.\{\rho_1\}R_1|b.\{\rho_2\}R_2$. In the explicit substitution semantics we can build the following inference tree:

$$\frac{c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{ff}} \theta}{c_1!c_2.\{\rho_1\}R_1|b.\{\rho_2\}R_2 \xrightarrow{\Rightarrow} \{\rho_1\}R_1|b'.\{\rho_2\}R_2}$$

and it holds that $\overline{\overline{S'}} = \overline{\overline{\{\rho_1\}R_1|b'.\{\rho_2\}R_2}} = P_1|b'.P_2 = P'$. The other cases can be arranged by applying first some congruence steps (cf. rule (Cng)).

(In-tt) Analogous to the previous case, observing in the conclusion that $\overline{\overline{\{\theta \circ \rho_2\}R_2}} = P_2\theta$.

(Case-tt) Assume that T consists of an application of rule (Case-tt), and consider an explicit process S such that $\overline{\overline{S}} = P$. It is either

- $S = \{\rho\}\mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(y)\colon R_1\ \mathsf{else}\ R_2$; or

- $S = \mathsf{case}\ (\rho e)\ \mathsf{of}\ \mathsf{some}(y)\colon \{\rho\}R_1\ \mathsf{else}\ \{\rho\}R_2$,

where it must be $\overline{\overline{\{\rho\}R_i}} = P_i$ for $i \in [1,2]$ in both cases. Again, the first case reduces to second via a directed congruence step, thus let us focus on $S = \mathsf{case}\ (\rho e)\ \mathsf{of}\ \mathsf{some}(y)\colon \{\rho\}R_1\ \mathsf{else}\ \{\rho\}R_2$. Notice that since $\overline{\overline{S}} = P$ and $e \rhd \mathsf{some}(c)$, it follows $(\rho e) = \mathsf{some}(c)$. hence, in the explicit substitution semantics we can build the following inference tree:

$$\mathsf{case}\ (\rho e)\ \mathsf{of}\ \mathsf{some}(y)\colon \{\rho\}R_1\ \mathsf{else}\ \{\rho\}R_2 \xrightarrow{\Rrightarrow} \{[c/y] \circ \rho\}R_1$$

and we conclude observing that $\overline{\overline{S'}} = \overline{\overline{\{[c/y] \circ \rho\}R_1}} = P_1[c/y]$.

(Case-ff) Analogous to the previous case.

(Rec) Assume that T consists of an application of rule (Rec), and consider an explicit process S such that $\overline{\overline{S}} = P$. Then, we have $S = \{\rho\}A(e_0)$, with $(\rho e_0) = e$, and we can therefore derive the following inference in the explicit substitution semantics:

$$\{\rho\}A(e_0) \xrightarrow{\Rrightarrow} \{[(\rho e_0)/x]\}P_0 \quad \text{if } A(x) \triangleq P_0$$

and we conclude observing that $\overline{\overline{S'}} = \overline{\overline{\{[(\rho e_0)/x]\}P_0}} = P_0[e/x]$.

Step. There are two remaining cases, corresponding to the composite rules of the reduction semantics of Table 4.4.

(Cng) Assume that T consists of an application of rule (Cng):

$$\frac{P_1 \equiv P_2 \quad P_2 \xrightarrow{\equiv} P_3 \quad P_3 \equiv P_4}{P_1 \xrightarrow{\equiv} P_4}$$

Consider an explicit process S_1 such that $\overline{\overline{S_1}} = P_1$. Observe that by Lemma A.2.4 there exist S_2, S_3, S_4 such that

$$P_i \equiv P_{i+1} \;\Rightarrow\; S_i \Rrightarrow S_{i+1} \wedge \overline{\overline{S_{i+1}}} = P_{i+1}$$

for $i \in \{1,3\}$. Moreover, by inductive hypothesis we have

$$P_2 \xrightarrow{\equiv} P_3 \;\Rightarrow\; \left(S_2 \xrightarrow{\Rrightarrow} S_3 \wedge \overline{\overline{S_3}} = P_3\right)$$

Therefore, we can build the following inference tree in the explicit substitution semantics:

$$\frac{S_1 \Rrightarrow S_2 \quad S_2 \xrightarrow{\Rrightarrow} S_3 \quad S_3 \Rrightarrow S_4}{S_1 \xrightarrow{\Rrightarrow} S_4}$$

and we conclude observing that $\overline{\overline{S_4}} = P_4$.

(Cnt) Assume that T consists of an application of rule (Cnt), where $P = C[P_1] \xrightarrow{\equiv} C[P_2] = P'$ is inferred thanks to $P_1 \xrightarrow{\equiv} P_2$. Consider an explicit process S such that $\overline{\overline{S}} = C[P_1]$. We proceed by induction on the structure of context C:

Basis. Assume that $C = [\,]$. Then, we have $P = C[P_1] = P_1 = \overline{\overline{S}}$, $P' = P_2$, and we conclude observing that by inductive hypothesis it holds that

$$P_1 \xrightarrow{\equiv} P_2 \wedge \overline{\overline{S}} = P_1 \;\Rightarrow\; \exists S' \,.\, \left(\overline{\overline{S'}} = P_2 \wedge S \xrightarrow{\Rrightarrow} S'\right)$$

Step. There are two cases to consider.

- Assume that $C = (\nu c)\, C_0[\,]$. Then, we have $P = C[P_1] = (\nu c)\, C_0[P_1] = \overline{\overline{S}}$ and $P' = (\nu c)\, C_0[P_2]$. Now, according to the definition of $\bar{\cdot}$, S is either $\{\rho\}(\nu c)\, C_0^S[P_1^S]$ with $\overline{\overline{\{\rho\}(\nu c)\, C_0^S[P_1^S]}} = C_0[P_1]$ or $(\nu c)\, S_0$ with $\overline{\overline{S_0}} = C_0[P_1]$. Moreover, notice that the former reduces to the latter by a congruence step, and thus we shall focus on $S = (\nu c)\, S_0$. Observe that the inductive hypothesis applies to the sub-process S_0 of S, leading to

  $$C_0[P_1] \xrightarrow{\equiv} C_0[P_2] \wedge \overline{\overline{S_0}} = C_0[P_1] \;\Rightarrow\; \exists S_0' \,.\, \left(\overline{\overline{S_0'}} = C_0[P_2] \wedge S_0 \xrightarrow{\Rrightarrow} S_0'\right)$$

  and we conclude by building the following inference tree:

  $$\frac{S_0 \xrightarrow{\Rrightarrow} S_0'}{S = (\nu c)\, S_0 \xrightarrow{\Rrightarrow} (\nu c)\, S_0' = S'}$$

  where $\overline{\overline{(\nu c)\, S_0'}} = C[P_2] = (\nu c)\, C'[P_2]$ holds by definition of $\bar{\cdot}$, for we have $\overline{\overline{S_0'}} = C'[P_2]$.
- The reasoning for the case $C = C'[\,]|Q$ (and for the symmetric right-parallel context) proceeds analogously. □

Appendix B

Proofs for Ch. 5

This appendix contains the proof of correctness of the availability analysis presented in § 5.4 with respect to an explicit substitution semantics in the style of § 4.5. We shall limit to present the key results needed to lift the correctness proof from the robustness to the availability analysis, the overall organisation of the proofs developing as in Appendix A.

The equivalence of the latter and the reduction semantics discussed in § 5.2 is too close to the developments of Appendix A.2 for a detailed discussion to be beneficial. We limit to observe here that the notation $\overline{\overline{S}}$ is updated so as to account for the syntax with qualified names and patterns. In particular, substitutions are applied to patterns in the first rule for the case construct. Corresponding results can be established for Theorem A.2.1 and Lemmata A.2.1,A.2.2,A.2.3,A.2.4,A.2.5,A.2.6.

B.1 Correctness of the Availability Analysis

The correctness proof exploits an explicit substitution semantics in the style of § 4.5. The notion of *explicit process* is seamlessly extended to the Quality Calculus with patterns as follows:

$$S \ ::= \ \{\rho\}P \mid (\nu c^{\tau})\, S \mid b.S \mid t_1!t_2.S \mid S_1|S_2 \mid \mathsf{case}\ e\ \mathsf{of}\ \mathsf{some}(p)\colon S_1\ \mathsf{else}\ S_2$$

As for the directed structural rules, in the first part of Table 4.6 we only need to replace restrictions with qualified restrictions. In the second part of the table, in charge of pushing substitutions to active processes, the following

Table B.1: The transition relation $\longrightarrow$ of the Quality Calculus with patterns with explicit substitutions.

$$\frac{c_1!v_2 \vdash b \to b' \quad b' ::_{\mathsf{ff}} \theta}{c_1!v_2.\{\rho_1\}P_1 \mid b.\{\rho_2\}P_2 \longrightarrow \{\rho_1\}P_1 \mid b'.\{\rho_2\}P_2} \quad \text{(In-ff)}$$

$$\frac{c_1!v_2 \vdash b \to b' \quad b' ::_{\mathsf{tt}} \theta}{c_1!v_2.\{\rho_1\}P_1 \mid b.\{\rho_2\}P_2 \longrightarrow \{\rho_1\}P_1 \mid \{\theta \circ \rho_2\}P_2} \quad \text{(In-tt)}$$

$$\frac{\vdash v \bowtie p : \sigma}{\mathsf{case\ some}(v)\ \mathsf{of\ some}(p) \colon \{\rho_1\}P_1\ \mathsf{else}\ \{\rho_1\}P_2 \longrightarrow \{\sigma \circ \rho_1\}P_1} \quad \text{(Match)}$$

$$\frac{\nvdash v \bowtie p}{\mathsf{case\ some}(v)\ \mathsf{of\ some}(p) \colon \{\rho_1\}P_1\ \mathsf{else}\ \{\rho_1\}P_2 \longrightarrow \{\rho_1\}P_2} \quad \text{(Mismatch)}$$

$$\mathsf{case\ none\ of\ some}(p) \colon \{\rho_2\}P_1\ \mathsf{else}\ \{\rho_2\}P_2 \longrightarrow \{\rho_2\}P_2 \quad \text{(Case-ff)}$$

$$\{\rho\}A(e) \longrightarrow \{[(\rho e)/x]\}P \quad \text{if } A(x) \triangleq P \quad \text{(Rec)}$$

$$\frac{S_1 \Rrightarrow S_2 \quad S_2 \longrightarrow S_3 \quad S_3 \Rrightarrow S_4}{S_1 \longrightarrow S_4} \quad \text{(Cng)} \qquad \frac{S \longrightarrow S'}{C[S] \longrightarrow C[S']} \quad \text{(Cnt)}$$

changes are needed: (*i*) qualified names replace names in rule (S-new); (*ii*) a pattern p replaces the variable y in the left-hand side of rule (S-case), and the substitution ρ is applied to p in the right-hand side of the rule; (*iii*) when applying substitutions to binders, like in rule (S-bin), also input patterns are involved now.

Finally, the explicit substitution semantics is given by the set of rules displayed in Table B.1, the main differences with respect to the basic calculus being the use of values for terms and rules (Match) and (Mismatch).

As for satisfiability-related notation, the concepts introduced in Appendix A needs to be lifted to first-order logic. Substitutions $\overline{\rho}$ shall now map variables $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ to the counter-part of expressions e and terms t in the logic, which will be elements of the types OpValues and Values. For the sake of simplifying the notation, we shall abuse the notation use the same syntax for expressions and terms on the calculus side and OpValues and Values on the logic side, retaining the notation $\overline{\rho}$ to denote lifting a substitution from the semantics of the calculus to an assignment in the logic. Moreover, the satisfaction relation $\models$ is lifted naturally to the new settings, where the satisfiability of quantifiers is sought with respect to the domain dom introduced in § 5.5.

The first result we need is lifting the correctness of the analysis of expressions

to patterns, i.e., producing a pattern-aware counter-part of Lemma A.1.1.

Lemma B.1.1

$$(1)\quad \forall e \forall \rho \forall p.\big(\,(\exists v \exists \sigma.\,((\rho e) = \mathsf{some}(v) \wedge \vdash v \bowtie p : \sigma) \wedge \vdash e \bowtie p : \psi_e) \Rightarrow\ \overline{\sigma \circ \rho} \models \psi_e\big)$$

$$(2)\quad \forall e \forall \rho \forall p.\big(\,(\exists v.\,((\rho e) = \mathsf{some}(v) \wedge \nvdash v \bowtie p) \wedge \vdash e \bowtie p : \psi_e)\ \Rightarrow\ \overline{\rho} \models \neg\psi_e\big)$$

$$(3)\quad \forall e \forall \rho \forall p.\big(\,((\rho e) = \mathsf{none} \wedge \vdash e \bowtie p : \psi_e)\ \Rightarrow\ \overline{\rho} \models \neg\psi_e\big)$$

Proof. By induction on the structure of expression e, with inner inductions on the structure of values v and patterns p. □

Then, we need to lift the correctness of the analysis of binders to binders with patterns, i.e., producing a pattern-aware counter-part of Lemma A.1.2.

Lemma B.1.2 *Let Θ, θ be substitutions from variables $x \in \mathcal{X}$ to constant optional values, and b be a binder. It holds:*

$$\forall \Theta \forall \theta \forall b\,\big((\Theta \gg b) ::_{\mathsf{tt}} \theta \wedge \vdash b \blacktriangleright \varphi_b, \psi_b\ \Rightarrow\ \overline{\theta} \models (\varphi_b \wedge \psi_b)\big)$$

Proof. By induction on the structure of binder b, observing that θ is constructed according to the semantics of $::_{\mathsf{tt}}$ and that ψ_b is built so as to comply with $\vdash v \bowtie p$, as shown in the previous lemma for the more complex case patterns. □

Finally, the notion of goodness has to be updated to the new settings. In the following, we denote Ξ_P a formula produced by the availability analysis for the program point just before process P, in analogy with the notation Φ_P introduced in § 4.6.3.

Definition B.1 (Goodness with patterns) Let S be an explicit process. S is *good*, denoted $\mathsf{good}(S)$, if

$$((\exists C.S = C[\{\rho\}P]) \Rightarrow \overline{\rho} \models \Xi_P) \wedge$$
$$((\exists C.S = C[t_1!t_2.\{\rho\}P]) \Rightarrow \overline{\rho} \models \Xi_P) \wedge$$
$$((\exists C \exists \sigma.S = C[\mathsf{case}\ \mathsf{some}(v)\ \mathsf{of}\ \mathsf{some}(p) \colon \{\rho\}P\ \mathsf{else}\ \{\rho\}P_2] \wedge \vdash v \bowtie p : \sigma) \Rightarrow \overline{\sigma \circ \rho} \models \Xi_P) \wedge$$
$$((\exists C.S = C[\mathsf{case}\ \mathsf{some}(v)\ \mathsf{of}\ \mathsf{some}(p) \colon \{\rho\}P\ \mathsf{else}\ \{\rho\}P_2] \wedge \nvdash v \bowtie p) \Rightarrow \overline{\rho} \models \Xi_P) \wedge$$
$$((\exists C.S = C[\mathsf{case}\ \mathsf{none}\ \mathsf{of}\ \mathsf{some}(p) \colon \{\rho\}P_1\ \mathsf{else}\ \{\rho\}P]) \Rightarrow \overline{\rho} \models \Xi_P) \wedge$$
$$(\forall \Theta \forall \theta.((\exists C.S = C[b.\{\rho\}P] \wedge (\Theta \gg b) ::_{\mathsf{tt}} \theta) \Rightarrow \overline{\theta \circ \rho} \models \Xi_P)).$$

We can now state the main correctness results.

Theorem B.1.1 (Correctness of the robustness analysis) *For all systems*

$$\begin{array}{ll} \mathsf{define} & A_1(x_1) \triangleq P_1 \\ & \vdots \\ & A_n(x_n) \triangleq P_n \\ \mathsf{in} & \{\mathsf{id}\}P_* \end{array}$$

it holds that

$$\forall S\,.\,(\{\mathsf{id}\}P_* \longrightarrow^* S \;\Rightarrow\; \mathsf{good}(S))$$

Proof. The proof is organised by induction on the length k of the derivation sequence $\{\mathsf{id}\}P_* \longrightarrow^* S$.

Basis. If $k = 0$, then it is $S = \{\mathsf{id}\}P_*$ and $\Xi_{P_*} = \mathsf{tt}$, from which the result follows since $\mathsf{id} \models \mathsf{tt}$.

Step. Assume that the result holds for $k \leq k_0$; we shall prove it for $k_0 + 1$. The whole derivation sequence can be written as

$$\{\mathsf{id}\}P_* \longrightarrow^{k_0} S' \longrightarrow S$$

The inductive hypothesis applies to the first k_0 steps of the derivation, leading to

$$\mathsf{good}(S')$$

Therefore, we shall now show

$$S' \longrightarrow S \;\wedge\; \mathsf{good}(S') \;\Rightarrow\; \mathsf{good}(S)$$

that is, the last derivation step in the sequence preserves the goodness of the system. The result follows directly from Lemma B.1.5. □

The technical lemmata on which the theorem rests establish that the structural congruence and the explicit substitution semantics of the calculus preserve goodness of explicit processes, starting from $\{\mathsf{id}\}P_*$ which is indeed good, as $\Xi_{P_*} = \mathsf{tt}$ and $\mathsf{id} \models \mathsf{tt}$.

Lemma B.1.3 ($\Rrightarrow$ preserves goodness) *For all explicit processes S and S' it holds that*

$$\mathsf{good}(S) \;\wedge\; S \Rrightarrow S' \;\Rightarrow\; \mathsf{good}(S')$$

Proof. The proof is organised by induction on the shape of the inference tree for the congruence step $S \Rrightarrow S'$. The detailed proof closely retraces the one of Lemma 4.6.1, the main differences consisting in the cases for rules (S-case) and (S-bin), which now rely on Lemmata B.1.1,B.1.2. □

The second central result, stated in Lemma B.1.5 below, shows that the explicit substitution semantics preserves goodness of processes. Before stating the lemma, let us accommodate the correctness of the judgement $c_1!c_2 \vdash b \to b'$ with respect to the availability analysis.

Lemma B.1.4 *Let Θ, θ be substitutions from variables $x \in \mathcal{X}$ to constant optional values, and b be a binder. It holds:*

$$\forall\Theta\forall\theta\ (\vdash b \blacktriangleright \varphi_b, \psi_b \wedge\ (\Theta \gg b) ::_{\mathsf{tt}} \theta \quad \Rightarrow \quad \overline{\theta} \models \varphi_b \wedge \psi_b)$$

$$\Downarrow$$

$$\forall\Theta\forall\theta\forall c\forall v\ (c!v \vdash b \to b'\ \wedge \vdash b' \blacktriangleright \varphi_{b'}, \psi_b \wedge (\Theta \gg b') ::_{\mathsf{tt}} \theta \quad \Rightarrow \quad \overline{\theta} \models \varphi_{b'} \wedge \psi_b)$$

Proof. The proof is analogous to the proof of Lemma A.1.6. □

Lemma B.1.5 ($\longrightarrow$ preserves goodness) *For all explicit processes S and S' it holds that*

$$\mathsf{good}(S) \wedge S \longrightarrow S' \Rightarrow\ \mathsf{good}(S')$$

Proof. The proof is organised by induction on the shape of the inference tree for the transition $S \longrightarrow S'$, exploiting Lemma B.1.3 for accommodating transitions of congruent processes. The detailed proof closely retraces the one of Lemma 4.6.2, the main differences consisting in the cases for rule (Case-tt), now replaced by (Match), (Mismatch), which rely on Lemma B.1.1, and for rules (In-ff), which now rely on Lemma B.1.2. □

Appendix C

Proofs for Ch. 7

This appendix discusses the correctness of the protection analysis of Ch. 7.

C.1 Correctness of the Protection Analysis

The correctness of the protection analysis with respect to the semantics of the calculus is formalised as follows:

$$\text{if} \quad P|Q \Longrightarrow^* C[^l P'] \quad \text{then} \quad \exists \mathcal{N} \in \mathsf{attack}(M^l) \text{ s.t. } \mathcal{N} \subseteq \mathsf{fc}(Q)$$

i.e., for all the executions in which Q can drive P to l, the analysis computes a set of channels $\mathcal{N} \in \mathit{Names}$ that under-approximates the knowledge required of Q.

Technically, it is convenient to organise a formal proof in two steps. First, if $P|Q$ reaches l then $P|H[\mathsf{fc}(Q)]$ reaches l, where process H is the hardest attacker possible and is parametrised on the knowledge of Q. H can be thought as the (infinite) process executing all possible actions on $\mathsf{fc}(Q)$, and the proof simply argues that whatever Q can, H can (Fact C.1.1). A similar approach is detailed in [NRH02].

Finally, the second step shows that if $P|H[\mathcal{N}']$ reaches l, then there must be a set $\mathcal{N} \in \mathsf{attack}(M^l)$ such that $\mathcal{N} \subseteq \mathcal{N}'$ (Theorem C.1.1). Observe that this formulation corresponds to the qualitative analysis of § 7.3. However, as $\mathsf{attack}(\mathsf{minimal}(M^l)) \subseteq \mathsf{attack}(M^l)$, the correctness of the quantitative analysis of § 7.4 follows as a particular case.

Definition C.1 (Hardest attacker) Let $\mathcal{N} = \{c_1, \ldots, c_n\}$ be a finite set of channels. $H[\mathcal{N}]$ is the process that does all possible sequence of output actions over channels in $\mathcal{N}$:

$$H[\mathcal{N}] \triangleq (\nu d)\,(!(c_1!d)) \,|\, \ldots \,|\, (!(c_n!d))$$

(where labels are of no use hence omitted).

In the definition we used a fresh name d as output term, but any name can be chosen as P cannot check the content of input variables. Observe that the channels in $\mathcal{N}$ might be used to trigger necessary outputs on other channels, according to the constraints in $P^l_{\Leftrightarrow}$.

Fact C.1.1 *Let P, P', Q be processes and C, C' contexts. It holds that*

$$\textit{if} \quad P|Q \Longrightarrow^* C[^l P'] \quad \textit{then} \quad P|H[\mathsf{fc}(Q)] \Longrightarrow^* C'[^l P']$$

As a matter of fact, the only blocking actions in P are inputs, and since the calculus is value-passing, the execution of P is driven exclusively by (i) the number of output actions Q performs, (ii) the channels over which they are executed, and (iii) their order. Now, for each channel $c \in \mathsf{fc}(Q)$, that is, for each channel known to Q, by construction $H[\mathsf{fc}(Q)]$ interleaves an arbitrary number of output on c, thus mimicking all the possible sequence of output actions on $\mathsf{fc}(Q)$, among which is the one performed by Q.

The main correctness result is phrased as follows. Since the semantics is value-passing, the proof does not present any particular obstacle, and therefore we limit to present its structure and major cases.

Theorem C.1.1 (Correctness of the protection analysis) *Let P, P' be processes, C a context, and $\mathcal{N} \in$ Names a set of channels. It holds that*

$$\textit{if} \quad P|H[\mathcal{N}] \Longrightarrow^* C[^l P'] \quad \textit{then} \quad \left(\exists \mathcal{N}' \,.\, \mathcal{N}' \in \mathsf{attack}(M^l) \wedge \mathcal{N}' \subseteq \mathcal{N}\right)$$

Proof sketch. The proof is organised by induction on the length k of the derivation sequence $P|H[\mathcal{N}] \Longrightarrow^* C[^l P']$.

Basis. If $k = 0$, then it is $P = C[^l P']$, from which $\emptyset \in \mathsf{attack}(M^l)$, for $\bar{l}$ is a fact in $P^l_{\Leftrightarrow}$, and thus it does not entail any channel literal to be tt. Since $\emptyset \subseteq \mathcal{N}$, for all set $\mathcal{N}$, the thesis follows.

Step. Assume $k = k_0 + 1$. The derivation sequence can be written as

$$P|H[\mathcal{N}] \Longrightarrow^{k_0} C''[^{l'} P''] \Longrightarrow C'[^l P']$$

for some context C'' and process P''. The inductive hypothesis applies to the first k_0 steps of the derivation: there exists $\mathcal{N}'' \in \mathsf{attack}(M^{l'})$ such that $\mathcal{N}'' \subseteq$

$\mathcal{N}$. Now, it suffices to show that the last step in the derivation sequence, leading to reaching l, preserves the inclusion relationship.

Observe that the last step $C''[P''] \Longrightarrow C'[^{l}P']$ is entailed by combining rule (Sys) with a transition $P'' \xrightarrow{\alpha} P'$, where we assume that the contexts C'', C' take care of hiding restrictions preceding P'' and parallel components of P'', P' that are not affected by the transition. As for the congruence step in the premise of rule (Sys), observe that the rewrite cannot produce inputs or outputs not already considered by the analysis, as the latter always assumes replications to be unfolded. To conclude, a formal proof requires an induction on the shape of the inference tree for the transition $P'' \xrightarrow{\alpha} P'$.

Let us comment upon the case of rule (In-tt), which is the most interesting and the only non-trivial. Assume that the binder b is a simple input $c?x$, passing which the label of interest is attained. Now, it is either $c \in \mathcal{N}''$, in which case we conclude $\mathcal{N}' = \mathcal{N}'' \subseteq \mathcal{N}$, or $c \notin \mathcal{N}''$. Again, we have two cases.

If there exists a subset of $\mathcal{N}''$ which can trigger another component of P to make an output on c, we again conclude $\mathcal{N}' = \mathcal{N}'' \subseteq \mathcal{N}$. Otherwise, we are in the case $\varphi \wedge \overline{c} \Leftrightarrow \overline{l}$ with $c \notin \mathcal{N}''$ and $\overline{c}$ not a consequence of the literals corresponding to $\mathcal{N}''$. In the set of constraints we have $g_c \vee \varphi' \Leftrightarrow \overline{c}$. It must then be either $c \in \mathcal{N}$, or $\mathcal{N}''' \subseteq \mathcal{N}$, where $\mathcal{N}'''$ satisfies φ', otherwise $\mathcal{H}[N]$ would not pass the input. If we look at models of $P^{l}_{\Leftrightarrow}$, we have that either g_c is $\mathtt{tt}$ or φ' evaluates to $\mathtt{tt}$ – in every model. In the first case we conclude $\mathcal{N}' = \mathcal{N}'' \cup \{c\} \subseteq \mathcal{N}$. In the latter $\mathcal{N}' = \mathcal{N}'' \cup \mathcal{N}^{iv} \subseteq \mathcal{N}$, with $\mathcal{N}^{iv} \subseteq \mathcal{N}'''$, because the least way of satisfying φ' by the analysis under-approximates the least way of satisfying φ' by the semantics.

The same reasoning applies to the case in which b is a quality binder, as formulae are computed according to the semantics of quality binders.

Appendix D

Properties of Attack Trees

This appendix contains some results that substantiate the procedure for generating attack trees discussed in Ch. 8.

D.1 Properties of $[\![P]\!]\mathsf{tt}$ and of $[\![l]\!]$

Lemma D.1.1 *Let P be a closed process in the Value-Passing Quality Calculus, and assume that each variable x and name c occurring in P is bound exactly once. Then, for any variable x in P, there exists exactly one formula $\varphi \Rightarrow \overline{x}$ in the translation $[\![P]\!]\mathsf{tt}$, and $\overline{x}$ does not occur in φ.*

Proof. By induction on the structure of processes. In particular, observe that in Table 7.3 a literal $\overline{x}$ is added to Φ only when a case clause is met, and by hypothesis $\overline{x}$ must previously appear in a binder, for processes are closed.

Let us discuss now the *complexity* of the translation given in Table 7.3. Let $\mathsf{size}(\mathcal{C})$ denote the number of literals occurring in a set of formulae $\mathcal{C}$, that is, $\mathsf{size}(\mathcal{C}) = \sum_{\varphi \in \mathcal{C}} (\mathsf{size}(\varphi))$, where $\mathsf{size}(\varphi)$ counts the literals in φ.

Lemma D.1.2 *Let P be a closed process in the Value-Passing Quality Calculus. Assuming that P contains n actions, then $\mathsf{size}([\![P]\!]\mathsf{tt}) = O(n^2)$.*

Proof. If P consists of n actions, $[\![P]\!]\mathsf{tt}$ consists of at most $O(n)$ formulae. More in detail, $[\![P]\!]\mathsf{tt}$ consists of $n_o + n_c + n_i + n_b$ formulae, n_o being the number of outputs in P, n_c the number of case clauses, n_i the number of simple inputs (including the ones occurring within quality binders), and n_b the number of

binders. The number of literals in a formula depends linearly on the number of actions preceding the label at which the formula is generated (cf. Table 7.3), hence the number of literals in $[\![P]\!]\mathsf{tt}$ is asymptotically bounded by n^2.

It is interesting to observe that from a theoretical point of view $O(n^2)$ is a precise bound to $\mathsf{size}([\![P]\!]\mathsf{tt})$. Consider the process IN_n that consists of n sequential inputs $c_1?x_1.\ldots.c_n?x_n$. The number of literals in $[\![IN_n]\!]\mathsf{tt}$ grows with

$$\begin{array}{rcl} \sum_{i=1}^{n}(2(i-1)+3) & = & \sum_{i=1}^{n}(2i+1) = \\ & = & n+2\sum_{i=1}^{n} i = n+2\frac{n(n+1)}{2} = \\ & = & n^2+2n \end{array}$$

where i records the number of literals in the hypothesis Φ, we have omitted counting the tt conjuncts, and we leverage the fact that an input generates two formulae whose size is $\mathsf{size}(\Phi)+2$ adding 1 literal to the hypothesis, from which the relation $2(i-1)+3$ is derived. Similarly, the translation of a process made of alternating inputs and case clauses would grow quadratically (with greater constants than IN_n).

Lemma D.1.3 *Let P be a closed process in the Value-Passing Quality Calculus, and assume that each variable x and name c occurring in P is bound exactly once. Then, for all labels l occurring in P, the formula $[\![l]\!]$ built according to the rules of Table 8.1 contains no literal $\overline{x}$. In particular, $[\![l]\!]$ only contains literals related to channels c.*

Proof. By induction on the number of steps in the unfolding of the generation of $[\![l]\!]$, according to the rules in Table 8.1.

Bibliography

[AAV95] AAVV. *Linda: User's Guide and Reference Manual*. Scientific Computing Associates, 1995. 53

[AB05] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM (JACM)*, 52(1):102–146, January 2005. 89

[ACG86] Sudhir Ahuja, Nicholas Carriero, and David Gelernter. Linda and Friends. *Computer*, 19(8):26–34, 1986. 53

[Ach09] Dimitris Achlioptas. Random Satisfiability. In *Handbook of Satisfiability*, volume 185, pages 245–270. IOS Press, 2009. 15

[AF01] Martín Abadi and Cedric Fournet. Mobile values, new names, and secure communication. In *ACM Symposium on Principles of Programming Languages (POPL '01)*, volume 36, pages 104–115. ACM, 2001. 9, 77, 89

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997. 76, 89

[AG98] Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, January 1998. 9

[AGG+05] Gul Agha, Carl Gunter, Michael Greenwald, Sanjeev Khanna, José Meseguer, Koushik Sen, and Prasanna Thati. Formal Modeling and Analysis of DoS Using Probabilistic Rewrite Theories. In *Workshop on Foundations of Computer Security (FCS'05)*, 2005. 20

[AIL05] Madhukar Anand, Zachary Ives, and Insup Lee. Quantifying Eavesdropping Vulnerability in Sensor Networks. In *2nd VLDB Workshop on Data Management for Sensor Networks (DMSN)*, pages 3–9. Wiley, 2005. 38

[Ama97] Roberto M. Amadio. An Asynchronous Model of Locality, Failure and Process Mobility. In *COORDINATION*, volume 1282 of *LNCS*, pages 374–391. Springer, 1997. 53

[Amo94] Edward Amoroso. *Fundamentals of Computer Security Technology.* Prentice-Hall, 1994. 98, 116

[AN04] Jari Arkko and Pekka Nikander. Weak Authentication: How to Authenticate Unknown Principals without Trusted Parties. In *Security Protocols, 10th International Workshop (2002)*, volume 2845 of *LNCS*, pages 5–19. Springer, 2004. 25

[And72] James P. Anderson. Information Security in a Multi-User Computer Environment. *Advances in Computers*, 12:1–36, 1972. 24

[ANL01] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-Resistant Authentication with Client Puzzles. In *8th International Workshop on Security Protocols*, volume 2133 of *LNCS*, pages 170–177. Springer, 2001. 21

[Aur01] Tuomas Aura. DOS-Resistant Authentication with Client Puzzles (Transcript of Discussion). In *8th International Workshop on Security Protocols*, volume 2133, pages 178–181. Springer, 2001. 17

[AWK02] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *9th ACM conference on Computer and Communications Security (CCS'02)*, pages 217–224, 2002. 135

[BAD14] Nikolaj Bjørner and Phan Anh-Dung. nuZ - Maximal Satisfaction with Z3, 2014. 117

[Bae05] J. C. M. Baeten. A brief history of process algebra. In *Theoretical Computer Science*, volume 335, pages 131–146, 2005. 9

[BAF08] Bruno Blanchet, Martín Abadi, and Cedric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008. 89

[Bal87] Robert W. Baldwin. *Rule Based Analysis of Computer Security.* PhD thesis, MIT, 1987. 135

[BB90] Gerard Berry and Gerard Boudol. The Chemical Abstract Machine. In *17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '90)*, pages 81–94. ACM, 1990. 43

[BBD+03] Lorenzo Bettini, Viviana Bono, Rocco De Nicola, GianLuigi Ferrari, Daniele Gorla, Michele Loreti, Eugenio Moggi, Rosario Pugliese, Emilio Tuosto, and Betti Venneri. The KLAIM Project: Theory and Practice. In *Global Computing. Programming Environments, Languages, Security, and Analysis of Systems, IST/FET International Workshop (GC'03)*, volume 2874 of *LNCS*, pages 88–150. Springer, 2003. 53, 140

[BC10] David Basin and Cas Cremers. Degrees of Security: Protocol Guarantees in the Face of Compromising Adversaries. In *Computer Science Logic*, pages 1–18. Springer, 2010. 27

[BDP02] Lorenzo Bettini, Rocco De Nicola, and Rosario Pugliese. KLAVA: a Java package for distributed and mobile applications. *Software Practice and Experience*, 32(14):1365–1394, 2002. 140

[BDP07] Stefano Bistarelli, Marco Dall'Aglio, and Pamela Peretti. Strategic Games on Defense Trees. In *Formal Aspects in Security and Trust (FAST'06)*, volume 4691 of *LNCS*, pages 1–15. Springer, 2007. 136

[Ber96] D. J. Bernstein. SYN cookies. cr.yp.to/syncookies.html (Accessed: August 2014), 1996. 21

[Ber04] Martin Berger. Basic Theory of Reduction Congruence for Two Timed Asynchronous pi-Calculi. In *CONCUR*, volume 3170 of *LNCS*, pages 115–130. Springer, 2004. 53

[Ber05] Martin Berger. An Interview with Robin Milner. In *Short Contributions from the Workshop on Algebraic Process Calculi: The First Twenty Five Years and Beyond (PA'05)*, BRICS Notes Series, pages 35–45, 2005. 8

[BGM+12] Michele Bugliesi, Lucia Gallina, Andrea Marin, Sabina Rossi, and Sardaouna Hamadou. Interference-Sensitive Preorders for MANETs. In *9th International Conference on Quantitative Evaluation of Systems (QEST 2012)*, pages 189–198. IEEE Computer Society, 2012. 90

[BH00] Martin Berger and Kohei Honda. The Two-Phase Commitment Protocol in an Extended pi-Calculus. *Electr. Notes Theor. Comput. Sci.*, 39(1):21–46, 2000. 53

[BH02] Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002. 109

[BHJ+11] Johannes Borgström, Suqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Pohjola, and Joachim Parrow. Broadcast Psi-calculi with an Application to Wireless Protocols. In *9th International Conference on Software Engineering and Formal Methods (SEFM 2011)*, volume 7041 of *LNCS*, pages 74–89. Springer, 2011. 89, 91

[BJPV11] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011. 9, 42

[BK86] J. A. Bergstra and Jan Willem Klop. Algebra of communicating processes. *Mathematics and Computer Science, CWI Monograph*, 1:89–138, 1986. 9

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*, volume 950. The MIT Press, 2008. 11

[Bla02] Bruno Blanchet. From Secrecy to Authenticity in Security Protocols. In *9th International Static Analysis Symposium (SAS'02)*, pages 342–359, Madrid, Spain, 2002. Springer Verlag. 77

[Bla09] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009. 77, 89, 116

[BLMW13] Fabrizio Biondi, Axel Legay, Pasquale Malacaria, and Andrzej Wąsowski. Quantifying Information Leakage of Randomized Protocols. In *14th International Conference Verification, Model Checking, and Abstract Interpretation (VMCAI'13)*, volume 7737 of *LNCS*, pages 68–87. Springer, 2013. 129

[BLT13] Fabrizio Biondi, Axel Legay, and Louis-marie Traonouez. QUAIL : A Quantitative Security Analyzer. In *25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 702–707. Springer, 2013. 130

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*, volume 145. Cambridge University Press, 1998. 75

[BRN04] Mikael Buchholtz, Hanne Riis Nielson, and Flemming Nielson. A Calculus for Control Flow Analysis of Security Protocols. *International Journal of Information Security*, 2(3-4):145–167, 2004. 57, 72, 89

[BST10] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard Version 2.0. Technical report, 2010. 67

[BWA94] Paul Butcher, Alan C. Wood, and Martin Atkins. Global synchronisation in Linda. *Concurrency - Practice and Experience*, 6(6):505–516, 1994. 53

[CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Principles of Progamming Languages (POPL'77)*, pages 238–252, 1977. 11

[CDE+11] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. Maude Manual (Version 2.6), 2011. 75, 78, 90

[CDM14] Marco Carbone, Ornela Dardha, and Fabrizio Montesi. Progress as Compositional Lock-Freedom. In *COORDINATION*, volume 8459 of *LNCS*, pages 49–64. Springer, 2014. 116

[CDP+11] Luís Caires, Rocco De Nicola, Rosario Pugliese, Vasco Thudichum Vasconcelos, and Gianluigi Zavattaro. Core Calculi for Service-Oriented Computing. In *Results of the SENSORIA Project*, pages 153–188. Springer, 2011. 9

[CFG+10] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico. Satisfiability Modulo the Theory of Costs: Foundations and Applications. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *LNCS*, pages 99–113, 2010. 116

[CGG05] Luca Cardelli, Andrew D. Gordon, and Giorgio Ghelli. Secrecy and Group Creation. *Information and Computation*, 196(2):127–155, 2005. 90

[CJI+09] Richard Chang, Guofei Jiang, Franjo Ivančić, Sriram Sankaranarayanan, and Vitaly Shmatikov. Inputs of coma: Static detection of denial-of-service vulnerabilities. In *Computer Security Foundations Symposium (CSF'09)*, pages 186–199. IEEE, 2009. 22

[Coo71] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *STOC '71 Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971. 13

[COP13] Amin Coja-Oghla and Konstantinos Panagiotou. Going after the K-SAT Threshold. In *45th ACM symposium on Theory of Computing (STOC'13)*, pages 705–714. ACM, 2013. 15

[DB12] Pierpaolo Degano and Andrea Bracciali. Process Calculi, Systems Biology and Artificial Chemistry. In *Handbook of Natural Computing*, pages 1836–1896. Springer, 2012. 8

[DDH72] O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare. *Structured Programming*. Academic Press Ltd., 1972. 12, 139

[DDK96] M. Dacier, Y. Deswarte, and M. Kaaniche. Models and tools for quantitative assessment of operational security. In *12th International Information Security Conference (IFIP/SEC'96)*, pages 177–186, 1996. 135

[DDMA12] Isil Dillig, Thomas Dillig, Kenneth L. McMillan, and Alex Aiken. Minimum Satisfying Assignments for SMT. In *Computer Aided Verification (CAV'12)*, volume 7358 of *LNCS*, pages 394–409. Springer, 2012. 117

[DM94] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994. 105, 110

[dMB] Leonardo de Moura and Nikolaj Bjørner. Z3 - a Tutorial. Technical report, Microsoft Research. 14

[dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3 : An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4963 of *LNCS*, pages 337–340, 2008. 14

[dMB11] Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011. 13

[DPVD12] Alessio Di Mauro, Davide Papini, Roberto Vigo, and Nicola Dragoni. Toward a Threat Model for Energy-Harvesting Wireless Sensor Networks. In *4th International Conference on Networked Digital Technologies (NDT 2012), International Workshop on Wireless Sensor Networks and their Applications*, volume 294 of *Communications in Computer and Information Science*, pages 289–301. Springer, 2012. v

[DY83] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983. 24

[Edd06] Wesley M. Eddy. Defenses Against TCP SYN Flooding Attacks. *The Internet Protocol Journal*, 9(4):2–16, 2006. 21

[EG04] Joost Engelfriet and Tjalling Gelsema. A new natural structural congruence in the pi-calculus with replication. *Acta Informatica*, 40:385–430, 2004. 42

[EMA+12] Jonas Eckhardt, Tobias Mühlbauer, Musab Alturki, José Meseguer, and Martin Wirsing. Stable Availability under Denial of Service Attacks through Formal Patterns. In *15th International Conference on Fundamental Approaches to Software Engineering (FASE'12)*, volume 7212 of *LNCS*, pages 78–93. Springer, 2012. 20

[EMM09] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties. In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design V*, pages 1–50. Springer Berlin / Heidelberg, 2009. 90

[FGS06] Gianluigi Ferrari, Roberto Guanciale, and Daniele Strollo. JSCL: a Middleware for Service Coordination. In *Formal Techniques for Distributed Systems (FORTE 2006)*, volume 4229 of *LNCS*, pages 46–60. Springer, 2006. 140

[FH05] Adrian Francalanza and Matthew Hennessy. A Theory of System Behaviour in the Presence of Node and Link Failures. In *CONCUR*, volume 3653 of *LNCS*, pages 368–382. Springer, 2005. 53

[FMQ94] GianLuigi Ferrari, Ugo Montanari, and Paola Quaglia. A Pi-Calculus with Explicit Substitutions: the Late Semantics. In *19th International Symposium Mathematical Foundations of Computer Science (MFCS'94)*, number L in LNCS, pages 342–351. Springer, 1994. 42

[GFM10] Fatemeh Ghassemi, Wan Fokkink, and Ali Movaghar. Equational Reasoning on Mobile Ad Hoc Networks. *Fundam. Inform.*, 105(4):375–415, 2010. 90

[GJ05] V. Griffith and M. Jakobsson. Messin' with Texas Deriving Mother's Maiden Names Using Public Records. In *Applied Cryptography and Network Security*, volume 3531 of *LNCS*, pages 91–103. Springer, 2005. 114

[GJM12] B. B. Gupta, R. C. Joshi, and M. Misra. Distributed Denial of Service Prevention Techniques. *International Journal of Computer and Electrical Engineering*, 2(2):268–276, 2012. 20

[Gli83] Virgil Gligor. A Note on the Denial-of-Service Problem. In *Proceedings of the 1988 Symposium on Security and Privacy*, pages 139–149. IEEE, 1983. 23, 24, 25

[Gli84] Virgil Gligor. A Note on Denial-of-Service in Operating Systems. *IEEE Transactions on Software Engineering*, 10(3):320–324, 1984. 23, 24, 25

[Gli86] Virgil Gligor. On Denial-of-Service in Computer Networks. In *Proceedings of the 2nd International Conference on Data Engineering*, pages 608–617. IEEE, 1986. 23, 24, 25

[GLP04] Deepak Garg, Akash Lal, and Sanjiva Prasad. Effective Chemistry for Synchrony and Asynchrony. In *IFIP TCS*, pages 479–492. Kluwer, 2004. 44

[GlPT07] Jean Goubault-larrecq, Catuscia Palamidessi, and Angelo Troina. A Probabilistic Applied Pi–Calculus. In *5th Asian Conference on Programming Languages and Systems*, pages 1–16, 2007. 90

[God07] Jens C. Godskesen. A Calculus for Mobile Ad Hoc Networks. In *Coordination Models and Languages*, pages 132–150, 2007. 91

[Gol11] Dieter Gollmann. *Computer Security*. Wiley, 3rd edition, 2011. 17

[GPV12] Marco Giunti, Catuscia Palamidessi, and Frank D. Valencia. Hide and New in the Pi-Calculus. In *Proceedings Combined 19th International Workshop on Expressiveness in Concurrency and 9th Workshop on Structured Operational Semantics (EXPRESS/SOS)*, volume 89, pages 65–79, 2012. 113

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd edition)*. Addison-Wesley, 2006. 8

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985. 8, 83

[IB02] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Network and Distributed System Security Symposium (NDSS'02)*, pages 79–86. The Internet Society, 2002. 20

[INR] INRIA. Tutorials for CADP, LNT, and LOTOS. http://cadp.inria.fr/tutorial/ (Accessed: October 2014). 140

[ISOa] ISO/IEC 15408. Common Criteria for Information Technology Security Evaluation. 19

[ISOb] ISO/IEC 7498-2. Information technology - Open Systems Interconnection - Security Architecture. 1, 19

[Ive80] Kenneth E. Iverson. Notation as a Tool of Thought. *Communications of the ACM*, 23(8):444–465, 1980. 9

[JB99] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Networks and Distributed Systems Security Symposium (NDSS'99)*, pages 151–165, 1999. 21

[JSW02] Somesh Jha, Oleg Sheyner, and Jeannette M. Wing. Two formal analyses of attack graphs. In *Computer Security Foundations Workshop (CSFW'02)*, pages 49–63, 2002. 135

[JW09] Jan Jürjens and Tjark Weber. Finite Models in FOL-Based Crypto-Protocol Verification. In *ARSPA-WITS*, volume 5511 of *LNCS*, pages 155–172. Springer, 2009. 68

[JW10] Aivo Jürgenson and Jan Willemson. Serial Model for Attack Tree Computations. In *Information, Security and Cryptology (ICISC'09)*, volume 5984 of *LNCS*, pages 118–128. Springer, 2010. 135

[KAv14] Aapo Kalliola, Tuomas Aura, and Sanja Šćepanović. Denial-of-Service Mitigation for Internet Services. In *19th Nordic Conference on Secure IT Systems (NordSec'14)*, volume 8788 of *LNCS*, pages 213–228. Springer, 2014. 20

[KK07] Israel Koren and C. Mani Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann, 2007. 18

[KMRS10] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. Foundations of Attacks-Defense Trees. In *7th International Workshop on Formal Aspects of Security and Trust (FAST'10)*, volume 6561 of *LNCS*, pages 80–95. Springer, 2010. 135

[KMS12] Barbara Kordy, Sjouke Mauw, and Patrick Schweitzer. Quantitative Questions on Attack-Defense Trees. In *15th International Conference on Information Security and Cryptology (ICISC'12)*, volume 7839 of *LNCS*, pages 49–64. Springer, 2012. 128, 136

[KP11] Dimitrios Kouzapas and Anna Philippou. A Process Calculus for Dynamic Networks. In *Formal Techniques for Distributed Systems (FMOODS/FORTE 2011)*, LNCS, pages 213–227. Springer, 2011. 90

[KPYK13] Seung-Hoon Kang, Keun-Young Park, Sang-Guun Yoo, and Juho Kim. DDoS avoidance strategy for service availability. *Cluster Computing*, 16:241–248, 2013. 22

[Lav] Sandra Laville. Anonymous cyber-attacks cost PayPal £3.5m, court told. In *The Guardian*. http://www.theguardian.com/technology/2012/nov/22/anonymous-cyber-attacks-paypal-court (November 22, 2012 - Accessed: August 2014). 20

[LD10] A. Legay and B. Delahaye. Statistical Model Checking : An Overview. *ArXiv e-prints*, 2010. 11, 20

[Lee08] Edward A Lee. Cyber Physical Systems : Design Challenges. In *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, 2008. 27

[Lip09] R. J. Lipton. *The P=NP Question and Gödel's Lost Letter*. Springer, 2009. 15

[LM03] Stéphane Lafrance and John Mullins. An Information Flow Method to Detect Denial of Service Vulnerabilities. *Journal of Universal Computer Science*, 9(11):1350–1369, 2003. 26

[LS10] Ivan Lanese and Davide Sangiorgi. An operational semantics for a calculus for wireless systems. *Theoretical Computer Science*, 411(19):1928–1948, 2010. 90

[MBS11] Massimo Merro, Francesco Ballardin, and Eleonora Sibilio. A timed calculus for wireless systems. *Theoretical Computer Science*, 412(47):6585–6611, 2011. 90

[MBZ+06] Vaibhav Mehta, Constantinos Bartzis, Haifeng Zhu, Edmund Clarke, and Jeannette Wing. Ranking Attack Graphs. In *9th International Symposium on Recent Advances in Intrusion Detection (RAID'06)*, volume 4219 of *LNCS*, pages 127–144, 2006. 135, 136

[Mea99] Catherine Meadows. A formal framework and evaluation method for network denial of service. In *Computer Security Foundations Workshop (CSFW'99)*, pages 4–13. IEEE, 1999. 25

[Mea01] Catherine Meadows. A cost-based framework for analysis of denial of service in networks. *Journal of Computer Security*, 9(1):143–164, 2001. 25, 116

[Mer09] Massimo Merro. An Observational Theory for Mobile Ad Hoc Networks. *Inf. Comput.*, 207(2):194–208, 2009. 90

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Springer, 1980. 8

[Mil90] Robin Milner. Functions as processes. In *Automata, Languages and Programming*, volume 443 of *LNCS*, pages 167–180. Springer, 1990. 43

[Mil93a] Jonathan K Millen. A Resource Allocation Model for Denial of Service Protection. *Journal of Computer Security*, 2:89–106, 1993. 25

[Mil93b] Milner. Elements of Interaction - Turing Award Lecture. *Communications of the ACM*, 36(1):78–89, 1993. 8

[Mil99] Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999. 8

[Mil06] Robin Milner. Turing, Computing and Communication. In *Interactive Computation*, pages 1–8. Springer, 2006. 1, 10, 25

[Mit] Mitre. Weak Password Recovery Mechanism for Forgotten Password. In *Common Weakness Enumeration*. http://cwe.mitre.org/data/definitions/640.html (Accessed: August 2014). 99

[MM12] Damiano Macedonio and Massimo Merro. A Semantic Analysis of Wireless Network Security Protocols. In *NASA Formal Methods - 4th International Symposium (NFM 2012)*, volume 7226 of *LNCS*, pages 403–417. Springer, 2012. 91

[MO06] Sjouke Mauw and Martijn Oostdijk. Foundations of Attack Trees. In *8th International Conference on Information Security and Cryptology (ICISC'05)*, volume 3935 of *LNCS*, pages 186–198. Springer, 2006. 135

[MOPV12] Narciso Martí-Oliet, Miguel Palomino, and Alberto Verdejo. Rewriting logic bibliography by topic: 1990-2011. *Journal of Logic and Algebraic Programming*, 81(7-8):782–815, 2012. 90

[MS05] Ajay Mahimkar and Vitaly Shmatikov. Game-based analysis of denial-of-service prevention protocols. In *Computer Security Foundations Workshop (CSFW'05)*, pages 287–301. IEEE, 2005. 22

[MTHM97] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997. 32

[MV09] Sebastian Mödersheim and Luca Viganò. Secure Pseudonymous Channels. In *14th European Symposium on Research in Computer Security (ESORICS'09)*, volume 5789 of *LNCS*, pages 337–354. Springer, 2009. 94

[MZ09] Sharad Malik and Lintao Zhang. Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76, 2009. 12

[NG09] Sebastian Nanz and Jens C. Godskesen. Mobility Models and Behavioural Equivalence for Wireless Networks. In *COORDINATION*, volume 5521 of *LNCS*, pages 106–122. Springer, 2009. 90

[NH06] Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2):203–227, 2006. 90, 91

[NO06] Robert Nieuwenhuis and Albert Oliveras. On SAT Modulo Theories and Optimization Problems. In *Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *LNCS*, pages 156–169, 2006. 116

[NRH99] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999. 11

[NRH02] Flemming Nielson, Hanne Riis Nielson, and René Rydhof Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283(2):381–418, 2002. 167

[OBM06] Xinming Ou, Wayne F. Boyer, and Miles a. McQueen. A scalable approach to attack graph generation. In *Computer and Communications Security (CCS'06)*, page 336, New York, New York, USA, 2006. ACM. 135

[OGKW08] L.J. Osterweil, C. Ghezzi, J. Kramer, and A.L. Wolf. Determining the Impact of Software Engineering Research on Practice. *Computer*, 41(3):39–49, 2008. 29

[OSW] Choosing and Using Security Questions Cheat Sheet. In *Open Web Application Security Project*. www.owasp.org (Accessed: October 2014). 100

[Par01] Joachim Parrow. An introduction to the pi-calculus. In *Handbook of Process Algebra*, chapter 8, pages 479–543. Elsevier, 2001. 42

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998. 116

[PHS03] Henrik Pilegaard, Michael Reichhardt Hansen, and Robin Sharp. An Approach to Analyzing Availability Properties of Security Protocols. *Nordic Journal of Computing*, 10(4):337–373, 2003. 26

[PNR08] Henrik Pilegaard, Flemming Nielson, and Hanne Riis Nielson. Pathway Analysis for BioAmbients. *Journal of Logic and Algebraic Programming*, 77:92–130, 2008. 44

[Pon12] Cyber Security on the Offense: A Study of IT Security Experts. Technical report, Ponemon Institute, 2012. 2, 3

[PPQ05] Davide Prandi, Corrado Priami, and Paola Quaglia. Process Calculi in a Biological Context. *Bullettin of the EACTS*, 85, 2005. 8

[Pra95] K.V.S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995. 90

[PS98] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Workshop on New security paradigms (NSPW'98)*, pages 71–79, 1998. 135

[Puh06] Frank Puhlmann. Why do we actually need the Pi-Calculus for Business Process Management. In *9th International Conference on Business Information Systems (BIS 2006)*, volume P-85 of *LNI*, pages 77–89. Gesellschaft für Informatik, 2006. 8

[Pul89] Geoffrey K. Pullum. Topic...Comment - The great Eskimo vocabulary hoax. *Natural Language and Linguistic Theory*, 7:275–281, 1989. 10

[PV98] Joachim Parrow and Björn Victor. The Fusion Calculus. In *Logic in Computer Science (LICS'98)*, pages 176–185. IEEE, 1998. 9

[Rab08] Ariel Rabkin. Personal Knowledge Questions for Fallback Authentication: Security Questions in the Era of Facebook. *Proceedings of the 4th Symposium on Usable Privacy and Security (SOUPS'08)*, pages 13–23, 2008. 114

[Ram02] Vijay Ramachandran. Analyzing DoS-Resistance of Protocols Using a Cost-Based Framework. Technical report, Yale University, 2002. 26

[RC11] Simona Ramanauskaite and Antanas Cenys. Taxonomy of DoS Attacks and Their Countermeasures. *Central European Journal of Computer Science*, 1(3):355–366, 2011. 19

[RH98] James Riely and Matthew Hennessy. A Typed Language for Distributed Mobile Processes (Extended Abstract). In *Principles of Progamming Languages (POPL'98)*, pages 378–390. ACM, 1998. 53

[RH01] James Riely and Matthew Hennessy. Distributed processes and location failures. *Theoretical Computer Science*, 266(1-2):693–735, 2001. 53

[RKT12] Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, 5(8):929–943, 2012. 135

[RN02] Hanne Riis Nielson and Flemming Nielson. Flow Logic : A Multi-paradigmatic Approach. In *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, volume 2566 of *LNCS*, pages 223–244, 2002. 11

[RN07] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications: an Appetizer*. Springer, 2007. 8

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 3rd edition, 2009. 15, 125, 126

[RN13a] Hanne Riis Nielson and Flemming Nielson. Probabilistic Analysis of the Quality Calculus. In *Formal Techniques for Distributed Systems (FMOODS/FORTE'13)*, volume 7892 of *LNCS*, pages 258–272. Springer, 2013. 140

[RN13b] Hanne Riis Nielson and Flemming Nielson. Safety versus Security in the Quality Calculus. In *Theories of Programming and Formal Methods*, volume 8051 of *LNCS*, pages 285–303. Springer, 2013. 140

[RNKP11] Hanne Riis Nielson, Flemming Nielson, Jörg Kreiker, and Henrik Pilegaard. From Explicit to Symbolic Types for Communication Protocols in CCS. In *Formal Modeling: Actors, Open Systems, Biological Systems*, volume 7000, pages 74–89. Springer, 2011. 44

[RNP12] Hanne Riis Nielson, Flemming Nielson, and Henrik Pilegaard. Flow Logic for Process Calculi. *ACM Computing Surveys*, 44(1):1–39, January 2012. 11

[RNV12] Hanne Riis Nielson, Flemming Nielson, and Roberto Vigo. A Calculus for Quality. In *9th International Symposium on Formal Aspects of Component Software (FACS'12)*, volume 7684 of *LNCS*, pages 188–204. Springer, 2012. v, 4, 31, 32, 48

[RSF+09] Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš, and Thomas Engel. Runtime Monitoring and Dynamic Reconfiguration for Intrusion Detection Systems. In *Recent Advances in Intrusion Detection*

(RAID'09), volume 5758 of *LNCS*, pages 61–80. Springer, 2009. 126, 135

[Sch89] Uwe Schöning. *Logic for computer scientists*. Birkhäuser Boston, 1989. 12

[Sch99] Bruce Schneier. Attack Trees. *Dr. Dobb's Journal*, 1999. 135

[SG10] Lei Song and Jens C. Godskesen. Probabilistic Mobility Models for Mobile and Wireless Networks. In *Theoretical Computer Science - 6th IFIP International Conference (TCS 2010)*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 86–100. Springer, 2010. 90

[SGNB06] J. Smith, J. M. González-Nieto, and C. Boyd. Modelling Denial of Service Attacks on JFK with Meadows's Cost-Based Framework. In *Australasian Workshops on Grid Computing and e-Research*, volume 54, pages 125–134. Australian Computer Society, 2006. 26

[SHJ+02] Oleg Sheyner, Joshua W. Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated Generation and Analysis of Attack Graphs. In *2002 IEEE Symposium on Security and Privacy*, pages 273–284, 2002. 135, 136

[SK97] K.E. Sirois and S.T. Kent. Securing the Nimrod routing architecture. In *Symposium on Network and Distributed System Security (NDSS'97)*, pages 74–84. IEEE, 1997. 25

[SLK10] Tao Shu, Sisi Liu, and Marwan Krunz. Secure Data Collection in Wireless Sensor Networks Using Randomized Dispersive Routes. *IEEE Transactions on Mobile Computing*, 9(7):941–954, 2010. 85

[SMS11] Karem A. Sakallah and Joao Marques-Silva. Anatomy and Empirical Evaluation of Modern SAT Solvers. *Bulletin of the EATCS*, 103:96–121, 2011. 15

[Som] Ravi Somaiya. Activists Say Web Assault for Assange Is Expanding. In *The New York Times*. http://www.nytimes.com/2010/12/11/world/europe/11anonymous.html (December 10, 2012 - Accessed: August 2014). 20

[SRS10] Anu Singh, C.R. Ramakrishnan, and Scott A. Smolka. A process calculus for Mobile Ad Hoc Networks. *Science of Computer Programming*, 75(6):440–469, 2010. 90

[STCE14] Sang C. Suh, U. John Tanik, John N. Carbone, and Abdullah Eroglu, editors. *Applied Cyber-Physical Systems*. Springer, 2014. 2, 27

[SW01] Davide Sangiorgi and David Walker. *The Pi-calculus - A Theory of Mobile Processes*. Cambridge University Press, 2001. 44

[SW04] Oleg Sheyner and Jeannette Wing. Tools for Generating and Analyzing Attack Graphs. In *2nd International Symposium on Formal Methods for Components and Objects (FMCO'03)*, volume 3188 of *LNCS*, pages 344–371, 2004. 135, 136

[SWYS11] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. A Survey of Cyber Physical Systems. In *International Conference on Wireless Communications and Signal Processing (WSCP '11)*, pages 1–6, Nanjing, China, 2011. IEEE Computer Society. 27

[TSMO04] Prasanna Thati, Koushik Sen, and Narciso Martí-Oliet. An Executable Specification of Asynchronous Pi-Calculus Semantics and May Testing in Maude 2.0. *Electronic Notes in Theoretical Computer Science*, 71:261–281, 2004. 90

[VCT+14] Roberto Vigo, Alessandro Celestini, Francesco Tiezzi, Rocco De Nicola, Flemming Nielson, and Hanne Riis Nielson. Trust-based Enforcement of Security Policies. In *Trustworthy Global Computing, 9th Symposium (TGC 2014)*, LNCS, page To appear. Springer, 2014. v, 53

[Vig12] Roberto Vigo. The Cyber-Physical Attacker. In *7th ERCIM/EWICS Workshop on Cyberphysical Systems*, volume 7613 of *LNCS*, pages 347–356. Springer, 2012. v, 4, 27, 114

[VMO02] Alberto Verdejo and Narciso Martí-Oliet. Implementing CCS in Maude 2. In *Workshop on Rewriting Logic and its Applications WRLA02*, volume 71 of *Electronic Notes in Theoretical Computer Science*, pages 282–300. Elsevier, 2002. 90

[VMO06] Alberto Verdejo and Narciso Martí-Oliet. Executable structural operational semantics in Maude. *Journal of Logic and Algebraic Programming*, 67(1-2):226–293, 2006. 90

[VNR13] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. Broadcast, Denial-of-Service, and Secure Communication. In *10th International Conference on integrated Formal Methods (iFM'13)*, volume 7940 of *LNCS*, pages 410–427, 2013. v, 5, 74, 132

[VNR14a] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. Automated Generation of Attack Trees. In *27th Computer Security Foundations Symposium (CSF'14)*, pages 337–350. IEEE, 2014. v, 5, 121

[VNR14b] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. Availability By Design. In *19th Nordic Conference on Secure IT Systems (NordSec'14)*, volume 8788 of *LNCS*, pages 277–278. Springer, 2014. v

[VNR14c] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. Uniform Protection for Multi-exposed Targets. In *34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE'14)*, volume 8461 of *LNCS*, pages 182–198. Springer, 2014. v, 2, 5, 95

[VYD12] Roberto Vigo, Ender Yüksel, and Carroline Dewi Puspa Kencana Ramli. Smart Grid Security A Smart Meter-Centric Perspective. In *20th Telecommunications Forum (TELFOR), 2012*, pages 127 –130, 2012. 61

[Wei99] Christoph Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *16th International Conference on Automated Deduction (CADE- 16)*, pages 314–328. Springer-Verlag, 1999. 116

[Wika] Wikipedia. Low Orbit Ion Cannon. http://en.wikipedia.org/wiki/Low_Orbit_Ion_Cannon (Accessed: August 2014). 20

[Wikb] Wikipedia. Timeline of events associated with Anonymous. http://en.wikipedia.org/wiki/Timeline_of_events_associated_with_Anonymous (Accessed: August 2014). 20

[Wir71] Niklaus Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, 1971. 139

[WJHF04] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New Client Puzzle Outsourcing Techniques for DoS Resistance. In *11th ACM conference on Computer and communications security (CCS'04)*, page 246, 2004. 21

[WNR14] Shuling Wang, Flemming Nielson, and Hanne Riis Nielson. Denial-of-Service Security Attack in the Continuous-Time World. In *34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE'14)*, volume 8461 of *LNCS*, pages 149–165. Springer, 2014. 140

[XRK08] Kun Xiao, Shangping Ren, and Kevin Kwiat. Retrofitting Cyber Physical Systems for Survivability through External Coordination. In *41st Hawaii International Conference on System Sciences*, pages 1–9, 2008. 27

[YG90] Che-Fn Yu and Virgil D. Gligor. A Specification and Verification Method for Preventing Denial of Service. *IEEE Transactions on Software Engineering*, 16(6):581–592, 1990. 24

[YPS04] Abraham Yaar, Adrian Perrig, and Dawn Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy (S&P 2004)*, pages 130–143, 2004. 20

[ZNR14] Kebin Zeng, Flemming Nielson, and Hanne Riis Nielson. The Stochastic Quality Calculus. In *COORDINATION*, volume 8459 of *LNCS*, pages 179–193. Springer, 2014. 140